LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 3. Nº 19
1250 PTAS.

ARGENTINA 9'50 $
CHILE 3000 $

Comunicaciones en UNIX

Lenguaje HTML
Instalación de servidores WWW

Resultado 1er Concurso de Demos e Infografía

LINUX: Device Drivers

Y además:

- DEMOS: Blur Motion
- SONIDO: Filtros digitales en tiempo real
- RAY TRACING

CURSO DE PROGRAMACIÓN VISUAL BASIC 4.0

8 413042 303299 00019

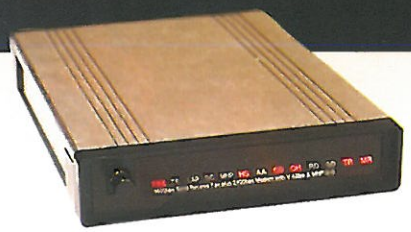
TOWER COMMUNICATIONS, S.R.L.

INTERNET

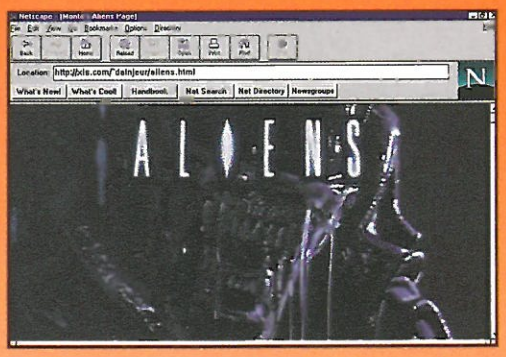
INTERNET

Directorio

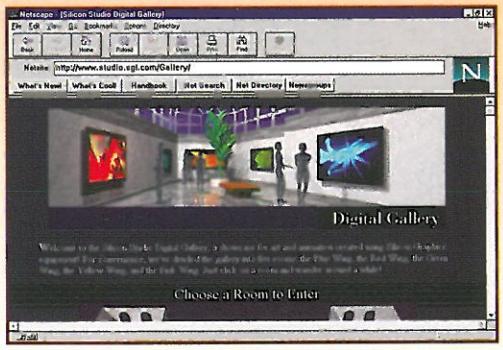
- Ciencia
- Cine
- Comics
- Compras
- Deportes
- Educación
- Filosofía
- Historia
- Humor
- Informática
- Libros
- Música
- Naturaleza
- Ocultismo
- Personajes
- Prensa
- Religión
- Revistas
- Salud
- Sexo
- Tecnología
- Televisión
- Viajes
- Vídeo-Juegos



ABETO



Por fin a la venta el 1er Directorio Internet



Por sólo 1.995 Ptas.

Solicita INTERNET DIRECTORIO enviando este cupón o llamando al teléfono (91) 661.42.11* de 9 a 14 y de 15:00 a 18:00 o por Fax: (91) 661.43.86

Deseo que me envíen: ☐ **INTERNET DIRECTORIO por 1995 ptas. + 250 ptas. de gastos de envío.**

Nombre y apellidos.....Domicilio.....
Población.....C.P .....Provincia.....
Teléfono .....F. de nacimiento.....Profesión.....

FORMA DE PAGO:

☐ Talón a ABETO EDITORIAL ☐ Contra-reembolso
☐ Giro Postal nº.....de fecha.....
☐ Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ CADUCA ☐☐ ☐☐

Firma ,

ABETO

Rellena este cupón y envíalo a:
ABETO EDITORIAL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

MARZO 1996. Número 19
SÓLO PROGRAMADORES es una publicación de Tower Communications

Editor
Antonio M. Ferrer Abelló
Director Editorial
Mario de Luis García
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Eduardo Toribio Pazos
Publicidad
Mª Eugenia González
Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta, Pedro Antón. Juan M. Martín, Luis Martín, José María Peco, José Carlos Remiro, Agustín Guillén, Álvaro Silgado, David Aparicio, Ignacio Cea, Daniel Navarro, Jorge del Río, Daniel Pérez, Jórge Sánchez, Pere Ridau, Jordi Freixenet, Enrique Torres

Maquetación
Fernando García Santamaría
Tratamiento de imagen
María Arce Giménez
Servicio Técnico
Óscar Rodríguez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Rosa Arroyo
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Aragoneses, 7
28100 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86
Filmación
Filma Dos S.L.
Impresión
G.D.B.
Distribución
SGEL

La revista SÓLO PROGRAMADORES no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792

CARTA DEL DIRECTOR

TIERRA Y LIBERTAD

La tierra es el lugar en el que vivimos y la libertad lo que hace posible disfrutar la tierra. Son tan importantes la una como la otra. Para los internautas, internet no es nada sin libertad. Para navegar por cualquier sitio, a cualquier hora y con cualquier edad. En Estados Unidos la cosa ha comenzado a cambiar. Algunos sectores del partido conservador han visto en Internet un nido de pornografía, perversión, foros de opinión sobre temas sexuales, xenofobia, etc... En definitiva, los temas sobre los cuales uno encontraría libros en cualquier biblioteca, al alcance de todos, pero metidos en la red. Esos *brillantes filósofos* argumentan que cualquier inocente niño puede acceder al WEB de PLAYBOY (uno de los más visitados del mundo) y allí, recibir información visual *perjudicial*. Y yo respeto esa forma de pensar. Lo que ocurre es que en una biblioteca que yo sepa, nadie le pone reparos a un menor para leer cualquier libro a cerca de la sexualidad o las *edificantes* novelas del Marqués de Sade, por poner un ejemplo. Pero no, estamos hablando de la informática, del *peligro* del siglo XXI, ¡Cuidado! Todas esas cosas, las redes, la información, hay que tenerlos controlados... Y ya han empezado. En EEUU ya hay empresas desarrollando *software censor* que, convenientemente instalado en las máquinas de los hogares, limite el acceso a los servidores considerados *peligrosos* o *poco recomendables* para los infantes de ciertas edades. Lo peor de todo, es que una idea que no es necesariamente mala puede degenerar en algo fuera de propósito. Imaginense una Comisión Censora de WEBS, ¿a qué empresa *con dos dedos de frente* le gustaría estar marcada como *poco recomendable?* Pongamos un ejemplo, una agencia matrimonial vía internet tiene en su sección de anuncios algunos ligeramente *picantes*, propios de la gente cuando busca un tipo determinado de relaciones. Lógicamente, la Comisión Censora clasificaría el centro, por dos anuncios que tuviera, como poco recomendable, dando además una palmadita en la espalda al intrépido censor que detectara los 2 anuncios malévolos entre los 200 normales. Resultado, la agencia, alertada por semejante escándalo, comenzaría a filtrar todo el material que le llegase pasándolo cada vez por un tamiz más severo. También habría agencias que al comprobar que tras unos meses, determinado tipo de personas no encuentran donde poner sus anuncios, crearían un servidor especializado en el que sólo se encontrara este tipo de material, separándolos del resto. Es lo malo de la censura, que alguien se arrogue el derecho a decidir por tí si algo es bueno o malo. Ahora Internet está de moda y las modas dan votos a los conservadores. Respeto cualquier postura pero recuerdo que en nuestro país cualquiera puede ir a una biblioteca y leer libros que seguramente estarían censurados en Internet de prosperar esta iniciativa. A nadie se le a ocurrido hacer un robot que vigile los libros, o los quioscos de prensa donde la edad no es obstáculo para adquirir *lo que sea*. Es más fácil pagar por un software determinado para acallar la conciencia de algún padre, más fácil eso que dar una educación comprensiva, pasar rato con la familia y, en último extremo, navegar con los infantes enseñándoles lo que es bueno y lo que no y, sobre todo, por qué. Lo otro, es una actitud propia de fariseos. El mejor software de protección no puede competir contra el hardware de esconder el cable de conexión o el teclado mismo si la cosa es tan grave... En mi opinión, Internet está bien como está, no creo que haya que cambiar su morfología ni su diseño, como mucho, vigilar de una forma coherente su uso, pero sin limitaciones ni Comisiones Censoras. Sólo espero que al final triunfe la cordura.

Bueno, bueno, dejemos por el momento la filosofía y vayamos a lo que interesa, la revista. Este mes, tras haber ampliado varias veces el cupo para la recepción de concursantes, publicamos los resultados del concurso de demos. El grupo de programación ganador de la *categoría reina*, demos de gran tamaño, ha sido HISPANIA. Nuestras felicidades desde aquí. El resto de los premiados podréis ver los resultados y la distribución de los premios en la página 78. En próximos números se analizarán en profundidad las técnicas de programación empleadas y se entrevistarán a los ganadores.

Por cierto, para los que a veces os cuesta encontrar la revista, el motivo es que AGOTAMOS algunos números, casualmente los que publicamos las últimas versiones del Linus Slackware que algunos *avispadillos* venden por 3 o 4.000 ptas. En SP somos así de espléndidos. La solución es el departamento de números atrasados de la editorial y estar atentos todos los meses.

Bueno, no molesto más, os dejo con este número de SP. Gracias por seguir con nosotros.

MARIO DE LUIS

SUMARIO

NÚMERO 19

6 CONTENIDO CD-ROM
Este mes continuamos con un gran nivel en cuanto a nuestro contenido en el CD. Lotus Gallery, una aplicaci—n multimedia para Windows. Además todos los fuentes de los art'culos.

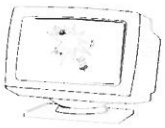
8 NOTICIAS
El espacio dedicado a informar al lector de las últimas novedades en el mundo de la informática y el comentario de los libros de interés.

12 WORLD WIDE WEB
La instalación de servidores WWW es una habilidad fundamental en el mercado laboral español en este momento, sin embargo, no supone ninguna dificultad especial.

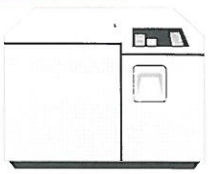
18 CURSO DE PROGRAMACIÓN
Un nuevo cap'tulo de esta entrega orientada a aquellos lectores con un nivel más bajo de conocimientos en programaci—n y a los que estén comenzando. Este mes Estructuras dinámicas de datos.

24 CURSO DE UNIX
Los principios básicos de uno de los mecanismos de distribuci—n de aplicaciones más usados sobre Unix y también disponible en Linux es el tema de esta entrega.

30 CÓMO HACER UNA DEMO
Blur Motion es el efecto que se consigue cuando una figura o sprite deja una estela a su paso. Veamos c—mo se programa dicho efecto.



34 GRANDES SISTEMAS
Manejar voluminosas cantidades de informaci—n, es uno de los principales problemas a abordar en los entornos de grandes sistemas. Este mes ficheros en MVS



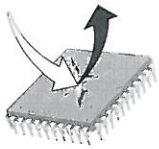
40 TÉCNICAS DE SONIDO
Todos estamos acostumbrados a ecualizar el sonido con unas barritas que subimos y bajamos, no son mas que filtros digitales en tiempo real.

45 SISTEMA OPERATIVO LINUX
Los device drivers son, posiblemente, la parte del sistema operativo más susceptible de ser implementada por un programadori

51 **PROGRAMACIÓN EN CLIPPER**
Clipper es uno de los lenguajes de programaci—n más solicitados profesionalmente hoy en d'a dentro del campo de las bases de datos.

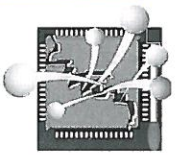
54 **PROGRAMACIÓN EN C++**
Un juego conversacional no es nada sin alguien que gobierne la interacci—n de todos sus personajes y objetos. Nos ocuparemos de este tema en esta última entrega.

57 **CREACIÓN DE UN COMPILADOR**
El éxito o fracaso de un lenguaje de programación tiene mucho que ver con la calidad de sus primeras implementaciones.



63 **CURSO DE RAY TRACING**
Sólo Programadores comienza un nuevo curso este mes: Visual Basic 4.0, nuestro objetivo acercar a los programadores más noveles a esta herramienta visual.

67 **CURSO DE VISUAL BASIC 4.0**
Sólo Programadores comienza un nuevo curso este mes: Visual Basic 4.0, nuestro objetivo acercar a los programadores más noveles a esta herramienta visual.



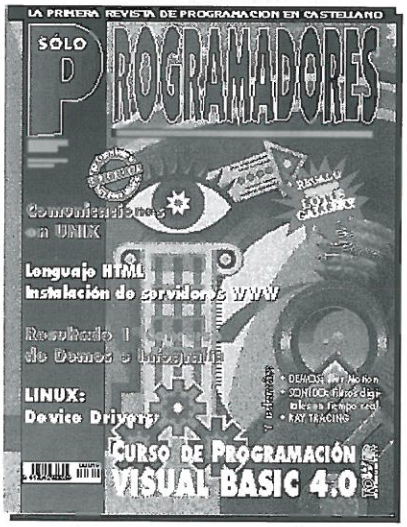
72 **CURSO DE VISUAL C++ 4.0**
Antes de continuar avanzando en el temario se realiza un breve paréntesis con el fin de estudiar un poco el entorno; programación en C del API Win32

78 **CONCURSO DE DEMOS**
Publicamos los resultados del primer concurso de Demos e Infograf'a organizado por Sólo Programadores. Elección dif'cil, pero prometemos más concursos.

79 **LINUX 3.00**
En el CD de Febrero, se ha ofrecido la última distribución Slackware de Linux, en su versión 3.00. Este mes comentaremos cómo poner a punto ciertos aspectos de la misma.

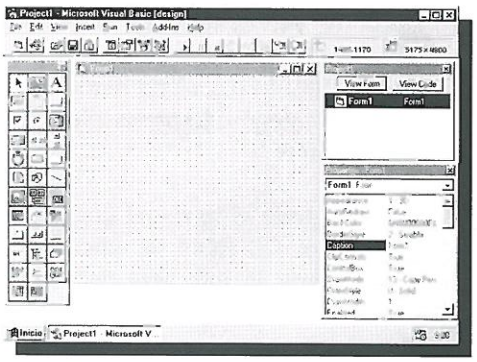
82 **CORREO DEL LECTOR**
Es el espacio dedicado a la resolución de los problemas surgidos a nuestros lectores en los diversos aspectos de la programación.

EN PORTADA



Visual Basic 4.0 es la nueva versión de la conocida herramienta visual de Microsoft. En Sólo Programadores estamos dispuestos a acercar el poder de estas herramientas visuales a nuestros lectores, consiguiendo que aquellos que sigan este curso sean capaces de desarrollar sus propias aplicaciones

DESTACADO



Ya tenemos ganadores de nuestro concurso de demos e infografía, por lo demás seguimos avanzando en nuestras secciones, y comenzamos otras que pensamos serán de gran interés como Visual Basic 4.0. Además un muy interesante artículo en la sección WWW sobre la instalación de servidores.

CONTENIDO DEL CD-ROM

LOTUS GALLERY

Este mes, junto con la revista, incluimos un CD-ROM de regalo con un contenido muy interesante. Se trata de Lotus Gallery, una aplicación multimedia interactiva para Windows, que muestra, de una manera espectacular, los últimos lanzamientos de la casa Lotus para este entorno

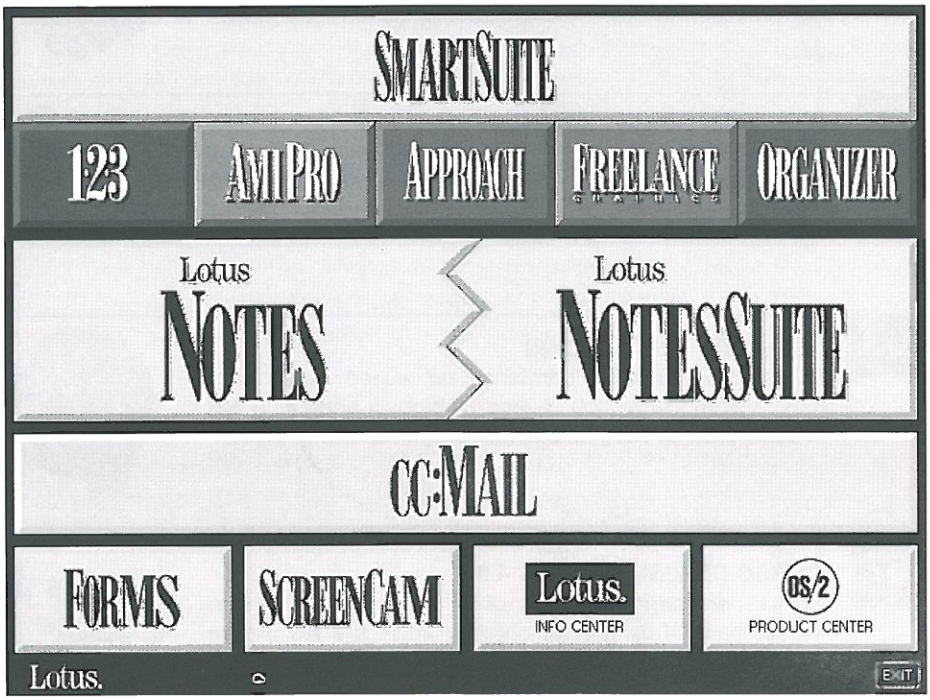
Además, contiene versiones de prueba de Lotus Notes, el nuevo estándar para trabajo en grupo, y Lotus SmartSuite 3.1, el famoso paquete integrado que compite mano a mano con el Office de Microsoft. La versión de prueba de Notes es idéntica a la completa, pero tiene desactivadas las funciones de comunicación, mientras que la versión de prueba de SmartSuite permite su uso completo pero sólo por un periodo de 45 días.

Y eso no es todo. También hay un bonito salvapantallas, una buena colección de cliparts de la librería Lotus SmartPics y una demo del programa Visio Express for Lotus SmartSuite.

NOTA: todo el material que se incluye en el CD-ROM se encuentra en inglés, por lo que será necesario un mínimo de conocimiento del mismo.

INSTALACIÓN DE LOTUS GALLERY

Desde Windows, seleccionar la opción Ejecutar en el menú Archivo y escribir x:install , siendo x la letra de la unidad de CD-ROM. La instalación requiere algo más de 2 MB libres en el disco duro. Una vez completada ésta, Lotus Gallery se ejecutará automáticamente, mostrando una bonita introducción y a continuación el menú principal.



Desde éste, se puede seleccionar el producto Lotus que nos interese y sobre el cual podremos ver una introducción multimedia, una demo, vídeos realizados con ScreenCam, documentos varios o podremos probarlo directamente.

Para el perfecto funcionamiento de la aplicación se recomienda el siguiente equipo mínimo:
- ordenador 486 33Mhz
- 8MB RAM
- 256 colores
- Tarjeta de sonido
- CD-ROM de doble velocidad.

INSTALACIÓN DE LOTUS SMARTSUITE

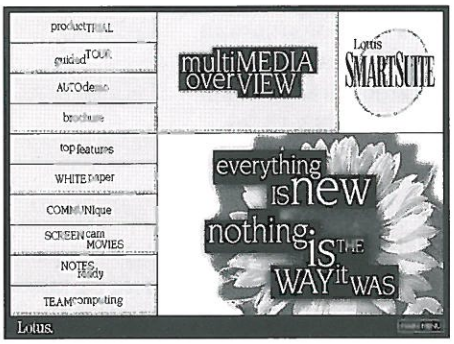
Una vez instalado Lotus Gallery, desde el menú principal, hacemos *click* sobre el botón correspondiente a SmartSuite, y en la siguiente pantalla seleccionamos la opción Product Trial. Hecho esto, accederemos a la instalación, donde se nos pedirá el directorio en el cual queremos instalar la aplicación, así como los componentes que deseamos instalar y si queremos que se ejecuten desde CD o desde el disco duro. Terminada la instalación, se habrá creado un nuevo grupo llamado Lotus Applications con los iconos de los componentes que hayamos instalado.

Como ya se ha comentado, ésta versión de Lotus SmartSuite es completamente operativa, pero sólo se puede utilizar 45 días, transcurridos los cuales se deshabilitará. Incluye los siguientes programas:
- Lotus 1-2-3 v5.0. Hoja de cálculo.
- Ami Pro v3.1. Procesador de textos.
- Approach v3.0. Base de datos relacional.
- Freelance Graphics v2.1. Presentaciones gráficas.
- Lotus Organizer v2.0. Agenda y organizador personal.
- Lotus ScreenCam v1.1. Capturador multimedia.

INSTALACIÓN DE LOTUS NOTES

Lotus Notes es un entorno para trabajo en grupo que ayuda a la gente a trabajar en equipo de una manera más eficiente. Con Notes, la gente puede trabajar en grupo sin importar plataforma, organización técnica o distancia geográfica. La clave para este trabajo en grupo es la compartición de información. La unidad básica de información en Notes son las Bases de Datos, con documentos y campos dentro de ellos.

Aunque Notes ha sido diseñado para trabajo en grupo, esta versión que ofrecemos (Lotus Notes Working Model) permite el uso individual. Es idéntica a la versión completa, pero con las funciones de comunicación deshabilitadas. Con ella, se puede



trabajar con ejemplos de bases de datos de Notes y empezar a desarrollar aplicaciones propias para Notes.

El procedimiento para instalar Lotus Notes es muy similar al de Lotus SmartSuite. Desde el menú principal de Lotus Gallery realizamos *click* sobre el botón Lotus Notes y a continuación, seleccionamos la opción Product Trial . A partir de aquí, no hay más que seguir las indicaciones que aparecen en pantalla.

Requerimientos mínimos:

- 8MB RAM
- 40MB libres en el disco duro, para el programa, los ejemplos y la documentación on-line.
- SVGA con 256 colores.

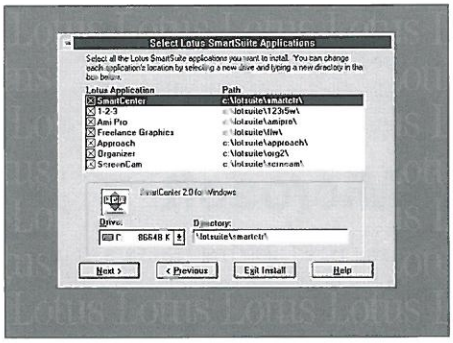
Terminada la instalación se habrá creado un nuevo grupo de programas llamado Notes Working Model CD, con varios iconos que dan acceso al programa, a una introducción y a una demostración.

INSTALACIÓN DE LOTUS SMARTPICS

Desde Windows, seleccionar la opción Ejecutar en el menú Archivo y escribir x:\fla\smrtpics\install , siendo x la letra de la unidad de CD-ROM. El espacio necesario son 700KB y se creará un nuevo icono llamado SmartPics.

INSTALACIÓN DEL SALVAPANTALLAS

Desde Windows, seleccionar la opción Ejecutar en el menú Archivo y escribir x:\lts\saver\setup , siendo x la letra de la unidad de CD-ROM. Esto copiará los archivos necesarios y creará un nuevo grupo de programas con



un icono llamado 3DPC, que es el programa de configuración del salvapantallas. Hay dos modelos distintos y merece la pena verlos los dos.

DEMO DE VISIO EXPRESS

Desde Windows, seleccionar la opción Ejecutar en el menú Archivo y escribir x:\lts\visio\vistutor , siendo x la letra de la unidad de CD-ROM.

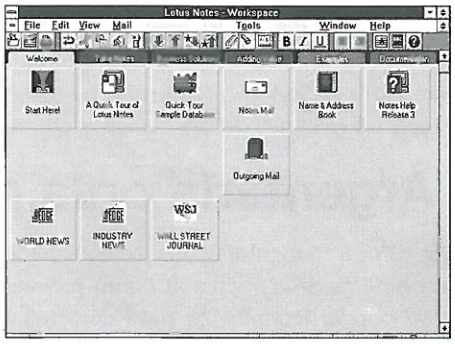
Visio Express for Lotus SmartSuite es un programa que permite confeccionar diagramas y gráficos de negocios para posteriormente incorporarlos a las aplicaciones de SmartSuite. Esta demo automática nos muestra cómo hacerlo.

MICROSOFT ROADMAP

En el CD-ROM del número anterior se entregó, junto con el Linux Slackware 3.0, una aplicación interactiva de Microsoft, llamada Roadmap, de la cual, por problemas de espacio en la revista, no se comentó nada. Pues bien, como más vale tarde que nunca, he aquí la breve reseña que debería haber aparecido en el pasado número.

Microsoft Roadmap es un sistema fácil e intuitivo para encontrar información completa sobre productos, programas y servicios de Microsoft. Esta información incluye los siguientes tópicos:

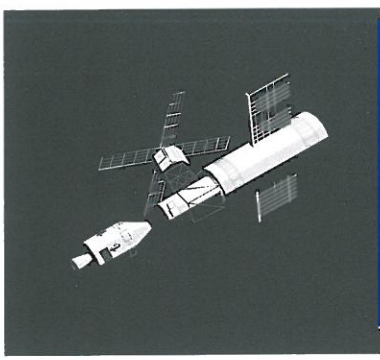
- herramientas visuales de desarrollo (Visual C++, Visual Basic, Visual Test, Visual SourceSafe, etc)
- desarrollo de aplicaciones a medida con MS Office
- bases de datos (MS Access, Visual FoxFro)
- sistemas operativos (Windows 95, Windows NT, BackOffice)
- estrategia de negocios
- especificaciones y estándares (OLE, ODBC, etc)
- programas y servicios



Además se facilitan direcciones de Internet y teléfonos de interés para solicitar mas información o pedir productos. Todo ello accesible mediante un cómodo interface de usuario.

En resumen es una herramienta de gran interés para todos los programadores pues en ella se halla un gran surtido de información para cada una de las aplicaciones que actualmente tiene Microsoft en el mercado. Especial interés puede tener para los lectores de Sólo Programadores toda la información relativa a Visual C++ 4.0, Visual Basic 4.0 y también Microsoft Fortran Powerstation una herramienta que se comentará en breve en la revista, la cual es capaz de trasladar la potencia de un gran sistema bajo Unix al mundo del PC.

En el directorio disk19 se encuentran los fuentes de los artículos correspondientes al número 19 de Sólo Programadores



SÓLO PROGRAMADORES

NOTICIAS

Bay Networks finaliza la adquisición de Xylogics

Bay Networks ha anunciado la finalización de la adquisición de Xylogics, proveedor líder de productos de acceso remoto de empresa. Xylogics funcionará como unidad independiente de Bay Networks y continuará ofreciendo a los usuarios y oficinas remotos el acceso a los recursos de red. La compañía Bay Networks ha intercambiado 8,7 millones de acciones comunes contra todas las acciones comunes de Xylogics. Además, ha reservado 1,6 millones de sus acciones comunes para emitir cuando se ejerzan las opciones previas de Xylogics.

En el pasado mes de septiembre ambas compañías firmaron un acuerdo definitivo de fusión por el cual Bay Networks iba a adquirir todas las acciones del capital de Xylogics intercambiando acciones comunes de Bay Networks, y según el cual Xylogics iba a convertirse en subsidiaria de Bay Networks.

Xylogics ha sido nombrada líder mundial en cuota de mercado de puertos de acceso remoto para la primera mitad de 1995 por las empresas de investigación de la industria informática International Data Corporation (IDC) y Dell'Oro Group. Según IDC, Xylogics alcanzó el 22,6 por ciento de la cuota de puertos servidores de acceso remoto basados en hardware.

Para más información:
Bay Networks
Ricardo Miranda-Naón
Tel: (91) 383 13 20

Argentaria opta por Microsoft

El Grupo Argentaria ha seleccionado recientemente a Microsoft como proveedor y socio tecnológico para desarrollar un programa de renovación tecnológica en sus 1.600 oficinas. Argentaria está utilizando los productos Microsoft Windows NT Server, Windows 95 y Office 95 como plataforma de desarrollo de las soluciones informáticas para sus sucursales.

Argentaria, con la colaboración y experiencia de Microsoft, desarrollará una solución que supondrá un gran salto cualitativo frente a los sistemas de que disponen en la actualidad las entidades financieras. El sistema realizará el proceso de las transacciones clásicas de la oficina y proporcionará al gestor comercial completa información sobre sus clientes y su actividad comercial: desarrollo de campañas, agenda comercial, resolución de consultas, nivel de comercialización de sus clientes, venta cruzada a clientes, etc. La facilidad de uso, la rapidez de respuesta y la capacidad de acceso a bases de datos centrales permitirá a los gestores comerciales de las oficinas proporcionar al cliente un servicio rápido y de calidad.

La implantación del sistema Windows NT en Argentaria se ha iniciado con el Banco Hipotecario, primer banco español en el sector hipotecario. La incorporación del resto de sucursales a los nuevos sistemas de Microsoft se completará durante el presente año.

Para más información:

Virginia Medina
Spanish Executive Information Service (S.E.I.S.)
Tel: (91) 308 43 15
Fax: (91) 310 38 76

Intel incrementa las prestaciones del Pentium a 166 MHz

Intel Corporation ha anunciado la inmediata disponibilidad de los procesadores Pentium a 150 y 166 MHz. Estos nuevos procesadores alcanzan altas prestaciones gracias a la tecnología avanzada de proceso de 0,35 micras a 3,3 voltios.

Según G. Carl Everett, vicepresidente senior de Desktop Products Group, los sistemas basados en procesadores Pentium rápidos seguirán siendo la elección ideal para la informática comercial y doméstica, mientras que el nuevo procesador Pentium Pro se ofrecerá en estaciones de trabajo, ordenadores de sobremesa y servidores avanzados.

Los dos nuevos procesadores son los primeros de la familia Pentium en alcanzar en nivel de prestaciones de 200 SPECint92. El procesador Pentium a 166 MHz proporciona prestaciones de 217,3 SPECint92, 4,76 SPECint95, 137,9 SPECfp92 y 3,37 SPECfp95. El procesador a 150 MHz proporciona prestaciones de 198,5 SPECint92, 4,27 SPECint95, 124,8 SPECfp92 y 3,04 SPECfp95.

En cantidades de 1.000 unidades el Pentium a 150 MHz cuesta 547 dólares y el procesador Pentium a 166 MHz tiene un precio de 749 dólares.

Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60
World Wide Web: http://www.intel.com

Olicom comercializa en España los modems de Lasat

Olicom ha anunciado que ya comercializa en España los productos de Lasat Communications A/S. Olicom ha suscrito el 35 por ciento de las acciones de Lasat, desarrollador y productor de modems de alta velocidad, sistemas avanzados de modem y equipos terminales RDSI (Red Digital de Servicios Integrados).

La gama de productos comercializada por Olicom es la siguiente:

- LASAT Unique 640i: modem RDSI externo con un tamaño equivalente a un disquette de 3 _ pulgadas. Tiene un precio de 55.980 pesetas.
- LASAT Safire 288 / Safire 144: Fax/Modem externos de diseño estilizado y exclusivo con un tamaño similar a un disquette de 3 _ pulgadas. El precio del Safire 288 es de 29.170 pesetas y el de Safire 144 es de 18.415 pesetas
- LASAT PC12 y PS12: adaptadores RDSI de uso interno en ordenadores PCs y PS/2, capaces de transmitir datos a 2 veces 64.000 bits/segundo, siendo posible transmitir 940 páginas de 1.000 caracteres en menos de 1 minuto.
- LASAT ST12: adaptador de terminal RDSI externo que puede ser utilizado para la comunicación síncrona o asíncrona. Tiene los mismos usos que el modelo anterior.
- LASAT Credit 288 / Credit 144: unos de los más pequeños Fax/Modem del mercado, con un tamaño de tarjeta de crédito, están diseñados para su uso en notebooks y palmtops con slots PC de tipo II. El precio de Credit 288 es de 30.950 pesetas y el de Credit 144 es de 25.780 pesetas.

Para más información:
Olicom Ibérica
Jacobo de Cal
Tel: (91) 345 72 76
Fax: (91) 457 20 79

Bay Networks y Concord Communications llegan a un acuerdo

Bay Networks y Concord Communications han anunciado la firma de un acuerdo cooperativo de desarrollo y marketing. Según los términos de este acuerdo, ambas compañías coordinarán las capacidades de análisis e informe de red automatizadas del producto Network Health de Concord Communicactions con la funcionalidad de análisis y monitorización basada en RMON/RMON 2 Advanced Analyzer de Bay Networks. Como consecuencia, los clientes podrán acceder fácilmente a una serie de información física, de red y de capa de aplicación utilizada para optimizar las prestaciones de la red, ejecutar la planificación de capacidad y manejar mejor los acuerdos a nivel de servicios.

Este nuevo acuerdo de desarrollo permitirá a Concord ampliar la familia Network Health para incluir Network Health-Bay Dialog, una nueva aplicación que analizará el tráfico de capa de red para identificar qué usuarios, departamentos y localizaciones están generando el volumen de red y utilizan los recursos en red.

Concord Communications entregará las nuevas aplicaciones Network Health en dos fases. Network Health-Bay Dialog estará disponible en el primer trimestre de 1996. La disponibilidad de informes de capa de aplicaciones se añadirá en la segunda mitad de 1996.

El primer precio de Network Health-LAN/WAN es de 9.995 dólares. El precio de Network Health-Bay Dialog empieza en 14.995 dólares para 25 interfaces.

Para más información:
Liliane Chinyavong
Abanico/Hot Line
Tel: (91) 594 43 53

Sybase presenta web.sql

Sybase Inc. acaba de anunciar el próximo lanzamiento de Sybase web.sql, un nuevo producto que ofrece enlaces abiertos a gran velocidad entre las bases de datos y los servidores Web de Internet. Sybase ha anunciado así mismo que integrará soporte para el lenguaje de programación Java en su familia de productos realizados para la red Internet.

Web.sql permite a las páginas Web acceder a los datos procedentes de bases de datos Sybase System 10 y Sybase System 11, así como a una muy amplia variedad de fuentes de datos. Mediante esta capacidad los desarrolladores de Internet e "Intranet" puedrán acceder de forma rápida y sencilla a los datos de las bases de datos existentes en una compañía que tengan una mínima codificación. Sybase web.sql ha sido diseñado para que sea posible trabajar tanto con servidores como con browsers Web del mercado.

Las divisiones de Sybase, New Media y Powersoft, ofrecerán herramientas de desarrollo basadas en el lenguaje de programación Java de Sun Microsystems. Además de esto, Sybase soportará también el acceso a su servidor de bases de datos de gran rendimiento SQL Server 11 a través de los interfaces de programas de aplicaciones (APIs) de Java. De esta manera se posibilita que las aplicaciones escritas con el mencionado lenguaje tengan la posibilidad de acceder a los datos corporativos a través de web.sql.

Por otra parte, el centro de formación Sybase Iberia /Powersoft España organizó una serie de cursos que abarcaron distintos temas relacionados con las tecnologías propias de ambas compañías, el pasado mes de Febrero.

Para más información:
Ludi García
Spanish Executive Information Service (S.E.I.S.)
Tel: (91) 308 43 15
Fax: (91) 310 38 76

NOVEDADES

Software AG presenta LightStorm

Software AG ha anunciado recientemente Natural LightStorm, el nuevo entorno de desarrollo profesional de desarrollo de aplicaciones cliente/servidor sobre plataformas Windows. Se trata de un entorno profesional que ofrece una total cobertura del ciclo de vida de un sistema de información apoyado en un repositorio, con soporte para grupos de trabajo o desarrolladores autónomos y que permite construir aplicaciones cliente/servidor de segunda generación.

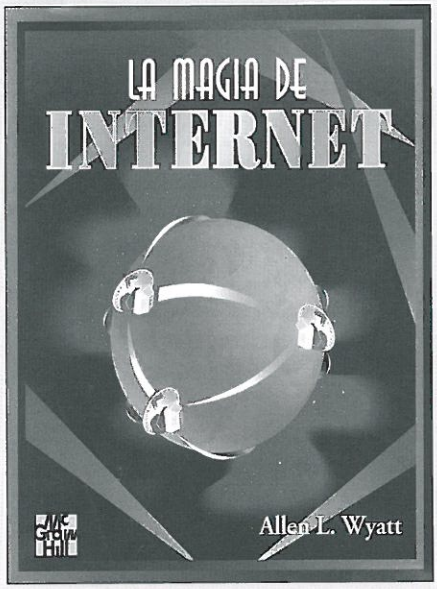
El objetivo principal de esta nueva solución es obtener un mayor valor añadido de aplicaciones heredadas en entornos cliente/servidor aplicando técnicas de reingeniería sobre los componentes individuales y realizando downsizing de las plataformas o incorporándolas dentro de nuevos desarrollos cliente/servidor.

Natural LightStorm soporta de manera integral el ciclo de vida de un sistema de información, desde la definición y análisis de los procesos empresariales, el diseño, la construcción y finalmente, la implantación en grupos de trabajo organizados. El repositorio de LightStorm almacena información tanto del negocio como de las aplicaciones que lo soportan.

LightStorm utiliza un sólo lenguaje para clientes y para servidores desarrollando aplicaciones cliente/servidor de segunda generación.

Para más información:
Software AG España
Ana Jérez
Tel: (91) 807 94 10
Fax: (91) 807 94 70

LIBROS



La magia de Internet

Ahora que la red Internet está tan de moda, muchos de los usuarios de ordenadores se encuentran perdidos ante la ingente cantidad de información que se les presenta. La magia de Internet es una obra dirigida a estos usuarios, en cuyas páginas se puede encontrar toda la información necesaria para aprender a navegar por la red de redes. El libro trata diversos temas como la conexión desde UNIX, DOS o MacIntosh, la búsqueda de información, la transferencia de ficheros y el correo electrónico. Está repleto de gráficos, cuadros y ejemplos, e incluye siete apéndices con información útil sobre Internet.

Editorial: McGraw-Hill
Autor: Allen L. Wyatt
472 páginas
Idioma: Castellano
Precio: Consultar

Todo lo que quiso saber sobre OS/2 WARP y no se atrevió a preguntar

Una de las tareas más complicadas con la que se enfrentan los usuarios de a pie es la correcta configuración del sistema operativo para sacar el máximo provecho a su ordenador. Esta obra de la editorial McGraw-Hill es un solucionario con los problemas más frecuentes que surgen en el manejo cotidiano del OS/2 Warp. Está compuesta por cientos de preguntas y respuestas organizadas en grupos como la superautopista de la información, los mensajes de error más frecuentes, la instalación y la impresión. Muchas de las respuestas incluyen gráficos, notas y sugerencias como complemento a la información proporcionada.

Editorial: McGraw-Hill
Autor: Kathy Ivens
314 páginas
Idioma: Castellano
Precio: Consultar

SI LOS PROGRAMADORES FUERAN ALBAÑILES

Uno de enero: Hoy me han llevado al solar por primera vez. La situación es perfecta: tiene el Metro a dos pasos y una cafetería enfrente donde sirven menú del día. El viejo bloque de pisos, al que va a sustituir nuestra nueva construcción, lleva un año al borde de la ruina. Mi propia empresa ha colocado varios puntales que, por el momento, han ido evitando que el caduco edificio reviente por sus múltiples grietas. La construcción de este megalito de ladrillo dio comienzo hace cinco años, y aunque los pisos superiores nunca llegaron a recibir el agua, la electricidad y el enfoscado de las paredes, en diez meses los cimientos ya se habían desplazado peligrosamente y las vigas presentaban peligrosas fisuras. La cansada torre de viviendas ya ha cumplido su propósito y ahora nosotros la conduciremos a una muerte dulce. Por supuesto, el viejo edificio no será demolido hasta después de construir y probar el nuevo, lo que nos deja poco espacio de maniobra; pero no vamos a dejar a todas esas familias en la calle durante la construción. De cualquier modo, los vecinos de la vieja y decadente estructura nos miran con recelo. Saben que el nuevo edificio tendrá viviendas más cómodas, pero algunos de los residentes no podrán costearlas. Ni sé qué va a ser de esta gente, ni es asunto mío. Llegan los primeros camiones de ladrillos.

Dos de enero: Me han presentado a Alberto, la persona a quien "voy a reportar". No me han dicho si es el capataz, el jefe de obra, el aparejador, o el arquitecto; sólo me han dicho que todo lo que tenga que "reportar", se lo "reporte" a él. Así que, por donde él diga, yo zaca zaca, como una locomotora. Ésa es la definición que me han dado de nuestra metodología. He buscado "reportar" en el diccionario, y no aparece.

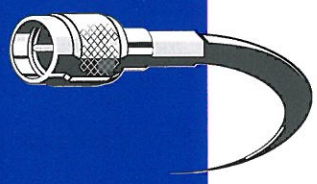
Seis de febrero: En algo más de un mes, hemos cavado medio metro de cimientos. Ayer Alberto nos dijo que empezáramos a poner ladrillos, porque el tiempo designado para la cimentación se había agotado hace dos semanas. No aceptó nuestras excusas de que las prometidas excavadoras aún no habían llegado, y que nos habíamos visto obligados a cavar con las paletas de enyesar. Un compañero se trajo una pala de cavar que guardaba de una obra anterior, y casi le echan por razones deontológicas. Según Alberto, lo que pasa es que frecuentamos demasiado la cafetería. El asunto se ha zanjado con un "hale, a levantar paredes y luego que cada palo aguante su vela". El trabajo sin planos es dificultoso. Los cimientos tienen una forma algo pintoresca. He pedido una plomada para que las paredes queden verticales, y he recibido improperios poniendo en duda mi masculinidad. Ya sé que Alberto no es el arquitecto, porque el arquitecto es un tal Ignacio. Pasó a supervisar la obra el otro día, aunque aún no hay nada que ver. Me han llegado rumores, aunque no son muy dignos de crédito, de que existen fotocopias de planos.

Doce de mayo: Anoche estuvimos hasta las siete de la mañana cubriendo con tablas y enmoquetando el espacio que algún día ocupará el despacho de la sexta planta, aunque el edificio no es aún más que una maraña de vigas de todos los tamaños y algunas paredes que habrá que tirar más tarde porque están en el sitio equivocado. Hemos traído baterías para los fluorescentes y unos muebles de caoba preciosos. Por suerte, todo estuvo a punto para la demo. Izamos al cliente con la grúa hasta su futuro despacho, y pudo contemplar la vista que se disfrutaría desde el emplazamiento. El viento hizo que la pared oeste, que dos de mis compañeros sujetaban con la espalda, se derrumbara con gran estruendo sobre la mesa de caoba en el peor momento. Gracias a Dios, el cliente fue comprensivo: esto pasa siempre en las demos, y él está curado de espanto, dijo mientras le sacudíamos el polvo del traje. Dice que el lunes que viene vendrá a probar las instalaciones sanitarias. Supliremos con cubos la inexistencia de tuberías.

Veintitrés de febrero: Han transcurrido casi catorce meses. Llevamos ya siete de retraso y el edificio no acaba de superar el estado de "casi terminado". Soy de los pocos albañiles que no ha cambiado de obra en este tiempo. Alberto está consumido por la zozobra, y se pasa el día en la cafetería trasegando Soberanos. El arquitecto no ha vuelto a pasar por aquí. Los rumores dicen que existieron unos planos, pero no eran de un bloque de pisos, sino de un polideportivo. Por lo visto, en las reuniones del comité de construcción se dijo que la filosofía era la misma, y que sólo harían falta modificaciones mínimas. Ahora comprendo por qué nos hicieron instalar aros de baloncesto en el hueco del ascensor. Siempre dije que acabaríamos teniendo que quitarlos o aquello no era un hueco de ascensor, que era cuestión de lógica. Alberto siempre me contestaba que no le viniera con tecnicismos. Estoy perdiendo la vocación de albañil. He decidido apuntarme por las tardes a un curso de informática, a ver si puedo cambiar de vida. Este oficio mío no es serio.

INSTALACIÓN DE SERVIDORES

Fernando J. Echevarrieta



La instalación de servidores WWW es una habilidad fundamental en el mercado laboral español en este momento y, sin embargo, no supone ninguna dificultad especial. Este artículo se centra en los aspectos que rodean la instalación y configuración de este tipo de servidor.

A lo largo de los últimos meses se ha estudiado la mayoría de las técnicas que puede utilizar un usuario cualquiera de Internet para presentar información en el WWW. Sin embargo, se ha llegado a un punto en el que para seguir avanzando, es necesario adentrarse en la programación CGI. Para ello, lo idóneo es convertirse en *webmaster*. Para lograr este fin, no será requisito disponer de ningún tipo de privilegios en ningún sistema, pero sí ralizar la instalación de un servidor de WWW, labor que puede realizar cualquier usuario. En el presente artículo se pesentarán las cuestiones técnicas que rodean la instalación de servidores WWW poniendo como ejemplo el del CERN, de libre distribución y modificación y adoptado en la actualidad por el *W3 Consortium*. Este servidor es, además, el único gratuito que cumple las funciones de proxy con caché, por lo que es idóneo para el aprendizaje.

Para aquellos lectores que no estén interesados en este aspecto, pero sí en otros como la programación CGI, el autor ha programado un sencillo script que genera una instalación básica del servidor mencionado para el sistema operativo Linux de forma automática. Aquellos lectores que también estén siguiendo el curso de UNIX encontrarán el script de gran interés al tratarse de un programa en bash que hace uso de casi todas las estructuras de control de la shell así como de funciones.

Aún existe otra alternativa si no tiene posibilidad o no desea instalar un servidor pero tiene intención de realizar programas CGI. Si este es su caso, deberá ponerse ya en contacto con el administrador de su servidor y plantearle el tema para que le autorice a escribir en un directorio reservado para CGIs. Hágalo cuanto antes para estar preparado para este tema el próximo mes.

EVOLUCIÓN DEL PANORAMA WEB

En el primer artículo de la serie, se presentó una panorámica general del estado del arte en tecnología WWW. Próximamente será necesario revisitarla ya que como se ha ido comentando este es uno de los terrenos de más rápida evolución en Internet. El draft relativo a tablas de HTML 3.0 no ha sufrido cambios sustanciales desde Noviembre hasta la fecha de escritura de este artículo y en Enero del '96 han aparecido nuevos drafts relativos al encadenamiento de hojas de estilo, con lo que parece que la idea se asienta; nuevos formatos gráficos como el *PNG*; inclusión de nuevas entidades en HTML y mecanismos de ampliación del HTTP. Microsoft ha entrado en escena con su *Internet Explorer* y ha anunciado que se podrá emplear Visual Basic para la programación del mismo. Netscape Corporation ha licenciado Java y desarrollado un lenguaje más simple, el *JavaScript* y parece que surge un convenio entre Spyglass Inc., Netscape Communications, Microsoft, IBM y SUN Microsystems para la inserción de objetos activos en documentos WWW.

Aunque es importante disponer de una visión global de la evolución de esta tecnología, es obvio que para construir un edificio es necesario comenzar por los cimientos. Así pues, en los anteriores artículos se presentó el lenguaje HTML desde sus aspectos más sencillos a las extensiones aún no estandarizadas. En el artículo en que se trataban las imágenes-mapa se men-

cionaba que Spyglass había desarrollado un sistema que permitía incluir estos mapas en los documentos HTML, y ser gestionados por el cliente. Las versiones beta del Netscape 2.0 incorporan unos *client-side imagemaps* que se corresponden exactamente con los definidos por Spyglass. Microsoft Internet Explorer también admite los mismos mapas en el cliente, por lo que parece que ésta será la tendencia futura y por supuesto, también será presentada en esta serie. En aquel mismo artículo se estudiaban los forms y se presentaba cómo todo form consta de dos partes: una parte HTML de presentación y otra en el servidor en forma de programa CGI para el proceso de la información obtenida. Del mismo modo, al hablar de actualización dinámica de documentos, el mes pasado se mencionaba la técnica *server-push*, manejada desde el servidor. Este es el punto límite en que conviene convertirse en *webmaster*.

SERVIDORES WWW

Un servidor no es otra cosa que un programa más o menos sencillo que se encuentra continuamente escuchando de la red. Cuando recibe una llamada establece, si procede, un diálogo, realiza una determinada acción y continúa a la escucha. Los servidores WWW "dialogan" mediante el protocolo HTTP, basado en TCP/IP (ver [Echeva1]) y escuchan, por convenio, en el puerto 80 (ver [Echeva2] y [Echeva3]). El funcionamiento de un servidor de WWW es bastante sencillo ya que consiste, fundamentalmente, en envíar al cliente los ficheros que éste último le solicita. Ficheros que pueden ser reales o resultado de la ejecución de un script (CGI) en ese momento.

La mayor parte de la complejidad del sistema reside en los clientes, que son los encargados de analizar la información recibida, para determinar si se trata de un formato conocido y representarla de la manera adecuada o si, por el contrario reciben un formato desconocido, en cuyo caso deberán lanzar la aplicación externa encargada de procesarlo o, simplemente, almacenar la información en disco. Pero las funciones de los clientes van más allá proporcionando una serie de facilidades al usuario, como guardar el estado de sus navegaciones para indicarle por donde ha pasado, etc.

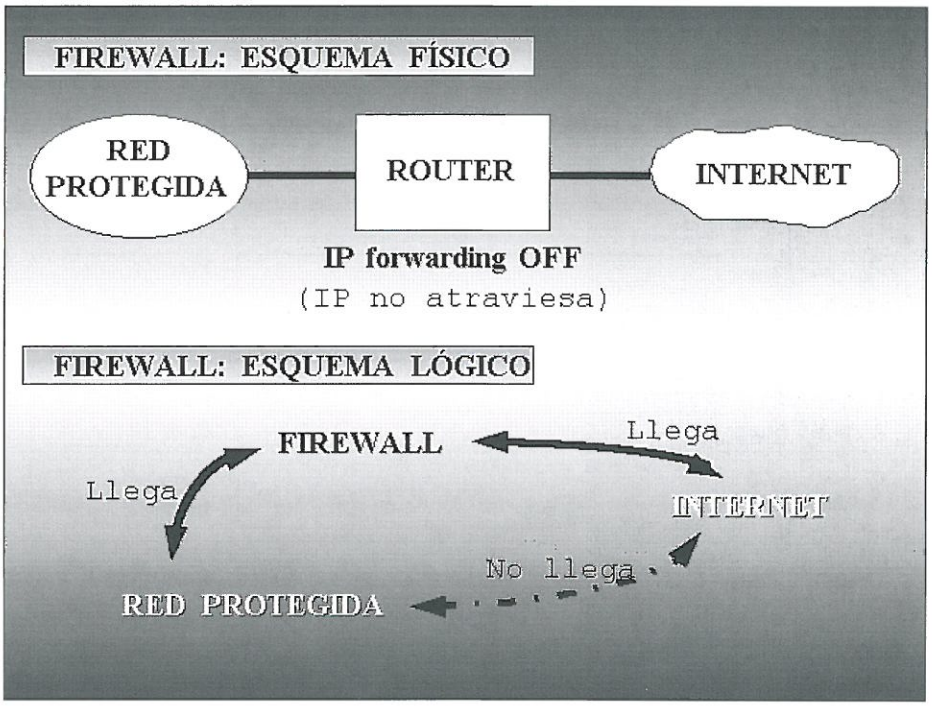


Figura 1. Estructura de un firewall.

Sin embargo, la mayoría de los servidores WWW proporcionan dos potentes mecanismos adicionales a su función original. El primero, relacionado con la seguridad en Internet: la posibilidad de actuar como *servidores proxy*. El segundo, relacionado con la eficiencia y disminución de costes: el actuar como *sistemas de caché*.

FIREWALLS

Uno de las principales cuestiones que surgen a la hora de facilitar información a través de la Internet o, simplemente, de acceder a la misma, es la seguridad. En la actualidad, el mecanismo de protección más utilizado es el denominado *firewall*, que no es otra cosa que un esquema lógico que protege una red determinada del resto de la Internet. El funcionamiento de un firewall es, en esencia, simple. Se trata de utilizar una máquina de la red a proteger como puente hacia la Internet como indica la figura 1. Configurando esta máquina para que no retransmita los paquetes IP a través de ella, no habrá forma directa de comunicar la red en cuestión con el resto de Internet o viceversa si no es a través de la máquina que actúa como firewall, que tendrá

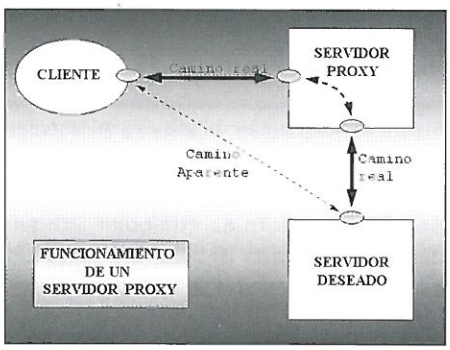


Figura 2. Funcionamiento de un servidor proxy.

acceso a ambas redes. De esta forma, cualquier acceso a la internet desde el interior, o a la red interna desde el exterior, se deberá hacer desde la máquina puente.

Por ejemplo, para realizar una conexión *telnet*, habrá que conectarse primero a la máquina que actúa como firewall y desde ésta, realizar el telnet a la máquina remota. Por poner otro ejemplo, para transmitir por *ftp* un fichero desde un nodo de internet a una máquina de la red interna será también necesario realizarlo en dos pasos. En un primer paso, se accedería a la máquina puente, por ejemplo a través de un telnet, y, desde ella, se realizaría la conexión ftp al nodo de internet. En un segundo paso habría que hacer llegar el fichero desde la máquina que

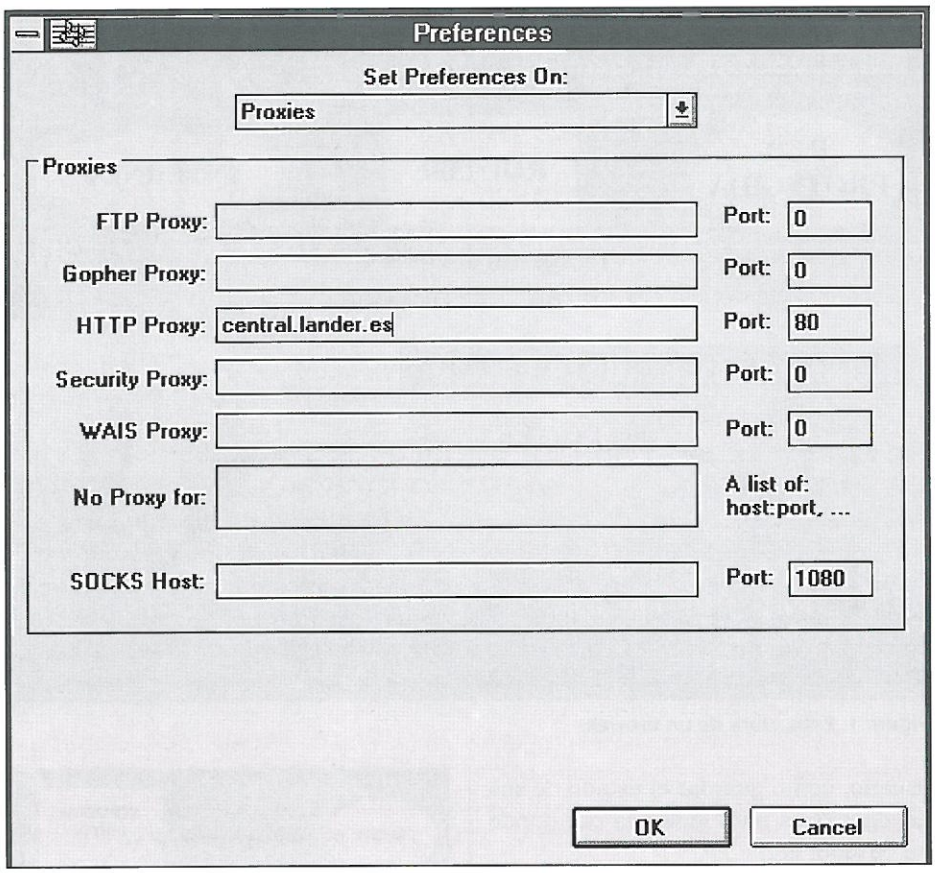


Figura 3. Asignación de proxies en Netscape.

actúa como firewall a la máquina interna. De esta forma, cualquier ataque desde el exterior deberá ser también realizado en dos pasos, siendo necesario atacar primeramente el firewall para ganar acceso al interior. Como contrapartida, el acceso desde el interior queda también bastante restringido, como se puede comprobar por los ejemplos. El problema llega a ser crítico si no se dispone de acceso a la máquina puente, con lo que se estaría aislado de la internet.

SERVIDORES PROXY

Para solucionar el problema de las restricciones de acceso a través de un firewall y también con el objeto de simplificar su gestión y minimizar el uso de recursos de las máquinas que hacen de puente, se ha diseñado una clase especial de servidores denominados *servidores proxy*. Un servidor proxy actúa como un puente entre un cliente interno y un servidor externo, como se puede apreciar en la figura 2. Por poner un ejemplo en el caso que nos ocupa, supongamos un usuario en la red interna sin acceso a la máquina que actúa de firewall. En principio, no podría acceder al WWW ya que su cliente tratará de realizar una conexión con el puerto 80 (por defecto) de un servidor en una máquina externa a la que no tiene acceso. Ahora bien, si en la máquina puente se coloca un servidor proxy de WWW el problema se habrá resuelto. Bastará indicar en la configuración del cliente que deberá utilizar esa máquina como proxy como se puede apreciar en la figura 3. De este modo, todas las peticiones se realizarán al servidor proxy indicado que a su vez, retransmitirá la petición al servidor deseado. Una vez obtenga el documento, lo devolverá al cliente como si fuera suyo.

CACHÉ EN SERVIDORES PROXY

La mayoría de los servidores proxy pueden, además, ser configurados como caché del servicio en cuestión. De este modo, se produce un gran ahorro de tiempo y de ancho de banda y, por tanto, de costes. Supóngase que en el escenario presenta en el apartado anterior, el servidor proxy dispone de un gran disco duro que almacena los documentos que pasan a través de él por un período limitado de tiempo. Si un usuario del interior intenta acceder a un documento, el servidor proxy comprueba si éste está en su cache. En caso afirmativo, lo servirá sin realizar una conexión con el exterior con el consiguiente ahorro en ancho de banda. El usuario, además, recibirá inmediatamente el documento, sin tener que esperar el tiempo de llegada de la petición al servidor remoto y el de vuelta del documento.

A este respecto, es importante diferenciar las ideas de actuación como proxy, relacionado con la seguridad y como caché, relacionado con la eficiencia. Un servidor que actúa como caché, siempre será un servidor proxy; pero un servidor proxy podrá actuar sin caché.

EL PROGRAMA DE INSTALACIÓN CERN_INST

Como se adelantaba en el comienzo del artículo, para aquellos lectores que no deseen "luchar" con la configuración de un servidor o para los que prefieran tener un servidor funcionando cuanto antes y cuando tengan más tiempo entrar en detalles, el autor ha realizado un programa de instalación del servidor del CERN que se adjunta con la revista. La distribución consta de dos ficheros:

cern3bin.tar: incluye la distribución de binarios para Linux de la versión 3.0 del servidor junto con algunos otros ficheros útiles al mismo.

cern_ins: el programa de instalación del autor que deberá ser renombrado a *cern_inst* en Linux.

El menú principal consta de tres opciones: 1. Instalación del servidor; 2. Generación de una nueva configuración y 3. Lanzamiento del servidor (figura 4).

CERN_INST: INSTALACIÓN

Al entrar en la opción de instalación el programa preguntará por el directorio en que se encuentra el fichero *cern3bin.tar* (figura 5) con la distribución y buscará, si no se le indica ninguno, en su directorio home.

Tras ello, se le preguntará por el directorio donde desea instalar el servidor así como los subdirectorios en los que se alojarán los binarios, pro-

gramas CGI, documentos HTML, logs del sistema y por el nombre del fichero de configuración (figura 6). Si desea mantener las opciones por defecto bastará con que pulse ENTER, lo que alojará todo el servidor bajo su directorio home. En caso de que desee instalarlo en otro sitio recuerde que deberá tener permisos en el directorio elegido.

El siguiente paso será la configuración básica del servidor (figura 7). Nuevamente, bastará pulsar ENTER para que todo funcione correctamente, aunque a continuación se procede a explicar los puntos de la configuración básica:

1. Puerto de Atención: Será el puerto por el que el servidor atenderá las llamadas. El puerto por convenio para el protocolo HTTP es es 80, sin embargo, se ha tomado como opción por defecto el 8080. Esto es así porque los puertos inferiores al 1024 se hallan reservados para root por el sistema operativo, por lo que se producirá un error si una aplicación de usuario intenta utilizarlos. El programa le permitirá seleccionar un puerto menor que 1024 únicamente si es root. En caso de elegir una opción diferente al 80, los clientes deberán indicar el puerto en sus URL's. Por ejemplo: *http://highland.dit.upm.es:8000*

2. UID del servidor: Especifica la identidad (permisos) con que se ejecutará el servidor y, por tanto, los procesos CGI que lance. Esta opción únicamente tiene sentido si el servidor lo ejecuta root, ya que si no, el sistema no permitirá el cambio de identidad. En cualquier caso, el programa de instalación no permite utilizar esta opción si al instalar el servidor no lo hace como root. En este punto la seguridad es crítica por lo que se le aconseja deje las opciones por defecto salvo que sea necesario cambiarlas.

2. GID del servidor: Especifica el grupo que adoptará el servidor. Remítase al apartado anterior.

Figura 4. Menú principal de cern_inst.

3. Directorios de usuario: A través de esta opción se le indica al servidor dónde debe buscar los documentos html que se especifican con una URL de la forma:

http://host.dominio/~usuario/documento.html

El servidor, en lugar de buscar en su árbol de directorios, cambiará al directorio home del usuario indicado y allí, entrará en el sudirectorio especificado por la opción para buscar el documento. Así por ejemplo, si se deja la opción por defecto, al especificar *http://www.dit.upm.es/~echeva/index.html*, se recuperará el fichero */home/echeva/public_html/index.html*. Si no va a instalar un servidor propio y desea publicar documentos a través del WWW deberá preguntar por este directorio a su webmaster.

4. Creación de logs: Si se contesta "SÍ", el servidor generará dos ficheros de log en el directorio que se escogió al efecto cada vez que se lance. Uno será para los errores del sistema y otro para trazar todas las conexiones que se realicen.

Tras realizar la configuración, el programa generará los directorios pertinentes, desempaquetará la distribución y alojará cada fichero en su directorio correspondiente y generará el fichero de configuración y una página de bienvenida personalizada para cada usuario (figura 8). Una vez terminado el proceso aparecerá una ventana con la opción de lanzar el servidor (figura 9). Si contesta afirmativamente podrá ya acceder con su browser a su nuevo servidor para observar la página de bienvenida.

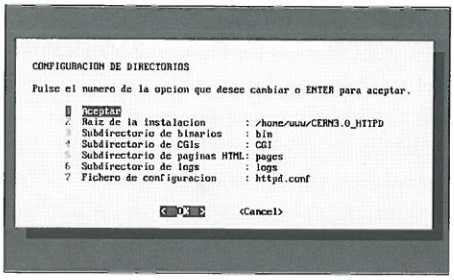


Figura 5

CERN_INST: NUEVA CONFIGURACIÓN Y LANZAMIENTO DEL SERVIDOR

Si al ejecutar el programa *cern_inst* se selecciona la opción de generar una nueva configuración, preguntará nuevamente por el directorio raíz de la instalación así como por el nombre del fichero de configuración. Se puede disponer de diferentes configuraciones en diferentes ficheros. Si se genera una nueva configuración sin cambiar el nombre del fichero, se guardará una copia de la antigua versión con extensión *.bak*.

El siguiente paso será nuevamente la pantalla de configuración del servidor vista en el punto anterior y la ventana de lanzamiento del mismo.

Si elige la tercera de las opciones del menú principal, se le preguntará ésta vez únicamente por la localización y nombre del fichero de configuración. Recuerde que puede disponer de varios. Tras ello y, previa confirmación, el servidor será lanzado.

Es importante destacar que el programa no "matará" los servidores que se encuentren lanzados. Tampoco comprobará si existe otro servidor escuchando en el puerto indicado. Por tanto, es responsabilidad del usuario el matar los servidores antes de lanzar otro en el mismo puerto. También hay que hacer notar que se pueden lanzar tantos servidores como se quiera siempre y cuando escuchen en un puerto diferente.

OPERACIÓN MANUAL DEL SERVIDOR

Es posible encontrar versiones precompiladas del servidor del CERN en *http://www.w3.org/* para multitud de

Un servidor proxy actúa como puente entre un cliente interno y un servidor externo

plataformas y sistemas operativos. Desgraciadamente, las únicas versiones del mismo que funcionan sobre PC son para UNIX. En cualquier caso, en el CD-ROM de la revista se han incluido dos servidores para Windows 3.x, pero no funcionarán si no dispone de un acceso real a red.

La existencia de versiones para tantas plataformas unido a sus características de servidor proxy con cache y a que es gratuito hacen del servidor del CERN la elección idónea para uso no profesional y, en ocasiones, también para el profesional.

El ejecutable, como la mayoría de los servidores de http tiene el nombre de *httpd*. Como fichero de configuración tomará por defecto el */etc/httpd.conf* pero es posible indicar otro fichero mediante la opción *-r*, lo que se hace necesario si un usuario distinto de root desea tener sus propias configuraciones. Por ejemplo:

httpd -r /home/echeva/httpd.conf

Antes de lanzarlo, deberá incluir en el fichero de configuración una línea de la forma:

Pass / /hypertext/WWW/**

que indicará al servidor que debe "mapear" todos los nombres de fichero añadiéndoles */hypertext/WWW/* (por ejemplo) delante y permitiendo el acceso a este directorio. Así, una petición como:

http://highland.dit.upm.es:8000/satelec.html

devolvería el documento:

/hypertext/WWW/satelec.html

Como se ha mapeado la raíz a un subdirectorio, lo que haya por encima de éste será inaccesible por el WWW.

El servidor, al ser lanzado, se pasa automáticamente a background y escribe su número identificador de proceso (PID) en el fichero *httpd-pid*. Así pues, una posible forma de matarlo sería:

kill `cat httpd-pid`

o de relanzarlo:

kill -HUP `cat httpd-pid`

aunque para ello, también se cuenta con la opción *-restart*:

httpd -restart

En cualquier caso, la experiencia del autor aconseja matar y volver a lanzar el servidor en lugar de rearrancarlo.

CONFIGURACIÓN BÁSICA DEL SERVIDOR DEL CERN

Los comentarios en el fichero de configuración comienzan por el carácter hash (#). En este fichero se puede incluir una serie de palabras clave, una en cada línea, que afectarán a su comportamiento (véase el generado por el programa *cern_inst* como ejemplo):

ServerRoot: Especifica cuál será el directorio "home" del servidor. Si no se indica nada más, tomará los iconos internos de un subdirectorio *icons* bajo éste.

HostName: Si el comando hostname produce en su sistema el nombre de la máquina sin el dominio, será necesario especificarlo aquí para el caso en que el servidor produzca redirecciones a sí mismo.

Port: El puerto de atención (ver cern_inst).

PidFile: Nombre del fichero en el que escribirá el número de proceso. Si se especifica *ServerRoot* lo creará en ese directorio. En caso contrario, en el */tmp*.

UserId: Usuario por defecto (ver cern_inst).

GroupId: Grupo por defecto (ver cern_inst).

IdentityCheck: Puede tomar los valores *On* y *Off*. Si se activa, el servidor llamará al demonio *ident* de la máquina en la que se encuentra el cliente para preguntar el identificador de usuario del cliente. Si consigue respuesta, lo incluirá en el fichero de log de accesos y lo pasará a los programas CGI mediante la variable de entorno *REMOTE_IDENT*. No es aconsejable utilizar esta opción ya que consume bastantes recursos y en algunos sistemas "se cuelga".

Welcome: Especifica el fichero de bienvenida por defecto. Puede aparecer varias veces estableciéndose un orden de preferencia. Por defecto, buscará en el siguiente orden: *Welcome.html*, *welcome.html* e *index.html*. En el caso de otro servidor muy extendido, el de NCSA, el fichero es siempre *index.html*.

Userdir: Especifica el subdirectorio de usuario (ver cern_inst).

REGLAS: MÁSCARAS Y REDIRECCIONES

Para el acceso a documentos o ejecución de CGIs, se definen una serie de reglas a través de palabras clave de la forma:

Regla <máscara> [resultado]

que se presentan a continuación:

Map <máscara> <resultado>

"Mapea" los nombres que se ajusten a la máscara según el resultado. Por ejemplo: *Map /* /hypertext/WWW/**. Las máscaras siempre pueden tener el carácter comodín *, pero los resultados sólo lo admiten si lo lleva su máscara.

Pass <máscara> [resultado]

Permite el acceso a los documentos que se correspondan con la máscara. Si se incluye máscara y resultado, engloba también la acción de *Map*. Por ejemplo:

Map / /hypertext/WWW/**
*Pass /**

será equivalente a

Pass / /hypertext/WWW*
Fail <máscara>

Deniega el acceso a los documentos que se ajusten a la máscara.

Exec <directorio1> <directorio2>

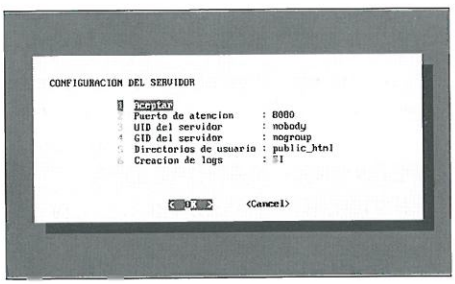


Figura 6

Figura 7

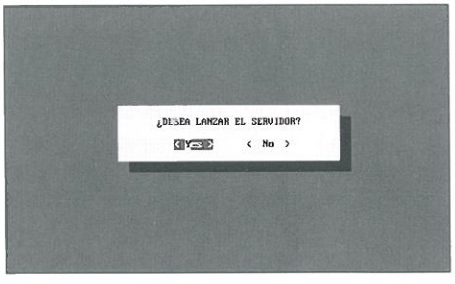


Figura 8

Realiza un "mapeo" especial. Cuando un URL se refiere al directorio1, el servidor entrará en el directorio2 y lanzará un programa que deberá cumplir el estándar CGI. Así, es habitual denominar a estos directorios como *htbin* o *cgi-bin*: Por ejemplo:

Exec /cgi-bin/ /home/www/cern/server/scripts/**

En este caso es obligatoria la existencia del carácter *.

Redirect <máscara> <resultado>

Redirige las peticiones de los documentos que se ajusten a la máscara, al resultado. El resultado debe ser un URL completo. Un ejemplo de esto podría ser:

Redirect /echeva/echeva.html http://highland.dit.upm.es:8000/cgi-bin/parser?/hypertext/WWW/echeva/echeva.html

que provoca que cualquier intento de acceder a la home page del autor en highland provoque la ejecución de un programa CGI que analizará el código HTML de la misma buscando una etiqueta definida por el autor para incluir la salida de ejecutables UNIX en su página. Estas técnicas serán estudiadas en los próximos artículos.

Las reglas se evalúan según aparecen en el fichero de configuración. Si una petición encaja exactamente con un *Map*, el resultado se aplica a las siguientes reglas. *Pass* y *Fail* terminan la búsqueda de reglas.

FUNCIONAMIENTO COMO PROXY

Para indicar al servidor que debe actuar como proxy sólo es necesario activar cada servicio mediante una instrucción *Pass* referida a una URL. Así:

*Pass http:**

le hará funcionar como proxy de WWW; pero también podrá actuar como proxy de otros servicios según indique la URL: *Pass gopher:**, *Pass ftp:** ...

También será posible indicar al servidor que, a su vez, debe actuar como cliente de otro servidor proxy para cada tipo de servicio. Esto se realiza mediante las palabras clave: *http_proxy <URL>*, *ftp_proxy <URL>*, *gopher_proxy <URL>*, *wais_proxy <URL>*.

Compruebe que no existen variables de entorno con estos nombres ya que podrían modificar el comportamiento del servidor, que también atiende a las mismas. La existencia de estas variables es posible debido a que es una forma de indicar a clientes de WWW que deben actuar a través de un proxy. En caso de que las variables existan deberá modificar el fichero de configuración si desea eludirlas.

Por último, es posible especificar una lista de lugares a los que debe acceder directamente aunque esté utilizando un proxy. Para ello, se define la palabra:

no_proxy <lista de lugares>

CONFIGURACIÓN DE LA CACHÉ

El manejo de caché permite un alto grado de configuración y está controlado por más de 20 palabras clave encargadas de fijar tamaños máximo y mínimo de la caché y de los documentos a almacenar, prioridades de entrada y salida, predicciones de caducidad, ajustes de horario y dimensiones de memoria de recolectores de basura, logs de acceso, etc. Las primitivas básicas son:

Caching <on\|off>

CacheSize tamaño en Mb (por defecto 5)

CacheRoot Directorio raíz de caché

CacheOnly máscara de URLs a los que se hará caché

NoCaching máscara de URLs a los que no se hará caché

CONCLUSIÓN

Mediante este artículo se ha presentado la forma de instalación y configuración de un servidor de WWW. Sin embargo, no se ha presentado de forma exhaustiva ni mucho menos la configuración del servidor del CERN cuyo manual consta de más de 60 páginas (http://www.w3.org/pub/WWW/Daemon). Aún dispone de otras características como las restricciones de acceso a usuarios, direcciones internet, o ambas, y a nivel de árbol, fichero o ambos, así como mecanismos de navegación de directorios. Por otra parte, lo habitual es actuar como proxy únicamente de un reducido número de máquinas de la propia organización, protección que no se ha presentado en el artículo. Aún así, el lector puede hacerse ya una idea más o menos clara de las características de los servidores de WWW y de la forma de configurarlas. Y lo que es más importante ya puede disponer de un servidor sobre el que aprender programación CGI.

BIBLIOGRAFÍA

- [Echeva1]. "Tecnología del WWW". Fernando J. Echevarrieta. Sólo Programadores, núm. 15
- [Echeva2]. "Arquitecturas Cliente-Servidor". Fernando J. Echevarrieta. Sólo Programadores, núm. 17.
- [Echeva3]. "Diseño de Aplicaciones Distribuídas sobre UNIX. SUN RPC: Llamada a Procedimiento Remoto". Fernando J. Echevarrieta. Sólo Programadores, núm 19.
- "World Wibe Web server software. Generated from the Hypertext", 2 de Junio de 1993. Berners Lee y otros.
- "NCSA httpd tutorial" http://hoohoo.ncsa.uiuc.edu/docs/tutorials/index.html
- "Windows httpd" http://www.city.net/win-httpd
- "How to put information on the WWW" http://www.w3.org/pub/WWW/Provider
- "Status of the W3C httpd" http://www.w3.org/pub/WWW/Daemon

NOTA SOBRE EL SERVIDOR FACILITADO

Se ha aludido a la versión 3.0 del servidor facilitado en todo momento como "servidor del CERN" o *CERN httpd*. La distribución realizada por SÓLO PROGRAMADORES se acoge al "Statment Concerning CERN W3 Software Release Into Public Domain", 3 de Mayo de 1993: "permission is granted for anyone to use, duplicate, modify and redistribute it". En posteriores releases de la versión 3.0, el copyright ha pasado a ser del MIT e INRIA pasándose a denominar *W3 Consortium httpd* y manteniéndose la licencia de libre distribución. Con la versión distribuida, la denominación es correcta.

CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica se anima al lector a ponerse en contacto con el autor mediante carta a la redacción de la revista o mediante:

E-mail internet: echeva@dit.upm.es

E-mail Compuserve: 100646,2456

WWW: http://highland.dit.upm.es:-8000.

ESTRUCTURAS DINÁMICAS DE DATOS

José C. Remiro

La combinación del tipo de datos puntero con estructuras de datos predefinidas por el lenguaje de programación permiten crear estructuras a gusto del programador y que pueden ajustarse a las necesidades del programa durante su ejecución

Todos los tipos de variables que se han presentado en artículos anteriores pueden denominarse estáticas. Las variables no son mas que una forma de direccionar de forma fácil para el programador una posición de memoria de un tamaño predeterminado por el lenguaje, dependiendo del tipo de datos que puede albergar dicha variable.

Cuando en un programa se hace referencia a una variable se está accediendo, bien para consultar bien para modificar, el contenido de dicha posición de memoria. Los contenidos de las variables pueden ser modificados en tiempo de ejecución pero no pueden ser destruidos, en el sentido en que no se puede liberar su memoria para destinarla a otro uso, ni tampoco puede ser modificado su tamaño. Así, si se ha declarado un array de 100 componentes éste no puede ser redimensionado mientras transcurre la ejecución del programa (algunos lenguajes como BASIC si permiten esta operación).

Hay algunas ocasiones en que no se conoce cuanta memoria se va a necesitar para ejecutar un programa. Por ejemplo, si se desea utilizar un algoritmo de ordenación utilizando un array, no siempre se ordenarán tantos elementos como componentes disponga el array, por lo que se estará derrochando memoria. Por el contrario, habrá ocasiones en las que se deseen ordenar mas elementos de los que disponga el array por lo que será necesario modificar el programa antes de ejecutarlo. En estas ocasiones, y otras similares, sería interesante disponer de estructuras que tengan en cuenta las necesidades de memoria adecuadas, liberando y obteniendo memoria según sea necesario. Esto es posible en los lenguajes de programación que dispongan de un tipo de datos denominado **puntero** y de procedimientos asociados para la gestión de memoria.

Punteros

Una variable de tipo puntero no contiene los datos de la misma forma en que los contiene una variable tal y como se conocen hasta ahora. En lugar de esto, una variable de tipo puntero almacena la dirección de memoria donde se pueden encontrar los datos a los que apunta. Así, debe haber una forma de poder diferenciar entre la dirección de memoria y los datos contenidos en dicha dirección.

Para declarar una variable de tipo puntero en el seudocódigo hasta ahora utilizado, se dispondrá el nombre de la variable, tras dos puntos se incluirá el carácter ≠ y a continuación el tipo de dato al que apunta (entero, real, etc.). Así, la declaración **apuntador: ≠ entero**, indica que se dispone de una variable apuntador que contiene la dirección de una posición de memoria que contiene un dato de tipo entero.

Ahora, habrá que distinguir entre el puntero y el contenido de la zona de memoria a la que apunta,de tal forma que, una variable puntero por sí sola únicamente comunicará una dirección, pero seguida del operador ≠ indicará que se desea acceder al contenido de la posición de memoria a la que apunta la variable puntero. Por ejemplo, **apuntador≠ = apuntador≠ + 3** incrementará en 3 el contenido de la posición de memoria a la que apunta la variable puntero **apuntador**. El efecto que tendrá **apuntador = apuntador + 3** será incrementar en 3 unidades la dirección de memoria

a la cual apuntaba **apuntador**, operación que no tiene sentido.

Como se había comentado con anterioridad, es necesario disponer de procedimientos para asignar y liberar la memoria a la que señala un determinado puntero, entre otras razones debido a que una variable puntero dispone de un tipo de datos al que apunta o direcciona, por lo que en cada caso el tamaño de la zona de memoria referenciada será diferente. Así, se considerarán los procedimientos, que pertenecerán al seudocódigo hasta ahora utilizado, **obtener_mem (puntero)** y **liberar_mem (puntero)**. El primer procedimiento asigna una dirección al puntero, donde puede encontrarse una posición de memoria adecuada para albergar un dato del mismo tipo que el del puntero. El procedimiento **liberar_mem**, permite liberar la zona de memoria asignada al puntero. Por último, se dispondrá de un valor especial de puntero que se designará por **nil**. Cuando un puntero tiene asignado dicho valor significa que al puntero no se le ha asignado aún ninguna dirección de memoria.

Estructuras dinámicas de datos

Los punteros no son importantes solamente por la forma en que referencian los datos, si no que combinados con estructuras ya conocidas pueden formar estructuras que crezcan o decrezcan durante la ejecución de un programa.

Para implementar este tipo de estructuras, se utilizarán los punteros como campos de un registro, al registro en su conjunto se le denominará nodo. El registro contendrá dos clases de información: una parte dedicada a almacenar la información a tratar y otra parte, representada por campos de tipo puntero, dedicada a gestionar la estructura de datos.

En el cuadro 1 se puede encontrar la definición de una lista, compuesta por una variable **l**, de tipo puntero y un nodo **nodolista**, representada por un registro. Este contiene dos campos: **inf** que contiene la información a tratar por la estructura y el campo **sig**, dedicado a concatenar varios de estos nodos (se pretende que el campo sig apunte al siguiente nodo de la estructura). Así, mediante las funciones **obtener_mem** y **liberar_mem** se pueden obtener nuevos nodos a los que apunte el puntero del nodo **sig**. Mediante la constante **nil** es posible conocer cuando se ha llegado al final de la estructura de datos.

Es posible incluir más de un campo de tipo puntero dentro de un nodo. El número de estos campos, indicará cuántos elementos son accesibles directamente desde un nodo. Así, podremos distinguir entre estructuras lineales de datos en las que a partir de un nodo se puede acceder como máximo al sucesor o al predecesor de un nodo (listas, pilas y colas) y las estructuras no lineales en las que a partir de un nodo se puede acceder a varios "sucesores" y en el que están implícitas la idea de bifurcación (árboles y grafos).

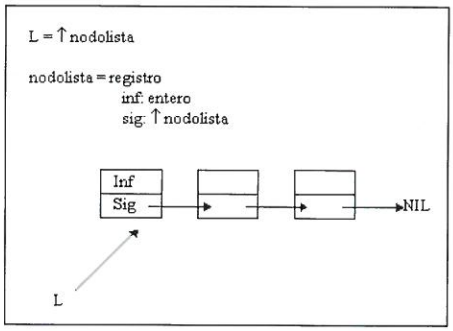
Es importante, como se verá mas adelante, que cuando se construya una estructura de datos dinámica se implementen una serie de procedimientos y funciones básicas para la utilización de dicha estructura, de tal forma que cualquier aplicación compleja pueda describirse a partir de las funciones y procedimientos básicos previamente construidos.

Listas lineales

La forma más simple de relacionar un conjunto de elementos es mediante una única lista. Los elementos, denominados nodos, se compondrán de dos clases de información: por un lado la información que se desea tratar y por otro un puntero que indicará la dirección del siguiente elemento de la lista.

La operación más sencilla que se puede realizar con una lista es la inserción de un elemento al comienzo de la lista. En el cuadro 2 se puede observar un algoritmo para crear una lista que contenga los primeros 15 números enteros positivos, de tal forma que la inserción de un nuevo elemento se producirá al principio de ésta.

Si se observa el conjunto de instrucciones que pertenecen a la instrucción **mientras**, se produce una llamada al procedimiento obtener_mem, que realiza una solicitud para que se asigne a la variable puntero aux una zona de memoria capaz de albergar un número



entero (información que se va a tratar) y la dirección del siguiente elemento (dato para poder gestionar la estructura de datos). Una vez obtenida esta zona de memoria, se incluyen en los campos del nodo al que apunta **aux** la información adecuada (recuérdese que el operador ≠ sobre un puntero indica el acceso a la zona de memoria a la que apunta). Como la inserción se realiza al principio de la lista, la dirección a la que debe apuntar el puntero del nodo al que apunta **aux** es la dirección del primer nodo de la lista, es decir, la dirección que contiene l, de ahí la asignación **aux** = l. A continuación hay que actualizar el puntero que indica el inicio de la estructura, esto es l, que deberá apuntar ahora a la dirección del primer nodo de la estructura

```
procedimiento crealista_15

l, aux : ↑nodolista
nodolista = registro
                inf: entero
                sig: ↑nodolista
 cont: entero
inicio
  l = nil
  cont = 0
  mientras cont < 15
     cont = cont + 1
     obtener_mem (aux)
     aux↑.inf = cont
     aux↑. sig = l
     l = aux
  fin mientras
fin
```

```
Tipo de datos que describe un nodo
nodolista = registro
              inf: cadena
              sig: ↑nodolista

funcion lista_vacia (l: ↑nodolista): logico
 inicio
  Si l = nil
    entonces
      lista_vacia = verdadero
    en otro caso
      lista_vacia = falso
 fin

procedimiento listar (l: ↑nodolista)
  aux: ↑nodolista
 inicio
  aux = l
  mientras aux <> nil
    escribir (aux↑.inf)
    aux = aux↑. sig
  fin mientras
fin
```

que ahora está siendo apuntado por el puntero **aux** (de ahí, la asignación final, l=aux).

Existen varios inconvenientes en la utilización de la anterior lista. En primer lugar, los elementos aparecerán en el orden inverso en que fueron generados. En segundo lugar, la localización de un elemento presente en la lista tendrá que pasar por inspeccionar en media la mitad de los elementos de la lista.

Para evitar estos inconvenientes, es posible mantener la lista ordenada. En los cuadros 3,4, 5 y 6 se encuentran los procedimientos y funciones necesarios para manipular una lista lineal ordenada. Se supone que la información que albergan los nodos son palabras que deben mantenerse ordenadas dentro de la lista. Se comentarán tan solo los procedimientos mas interesantes.

La función **lista_vacia** comprueba si la lista introducida como parámetro es la lista vacía o no. La lista estará vacía cuando el puntero que apunta al primer nodo tiene el valor **nil**.

El procedimiento **listar** recorre la lista visualizando la información contenida en cada uno de sus nodos. El puntero **aux** se inicializa con el mismo valor al que apunta el puntero que señala al primer nodo de la lista. Tras visualizar el contenido de dicho nodo, pasa a apuntar al siguiente nodo de la

```
procedimiento insertar ( var l: ↑nodolista; palabra: cadena)
  aux, anterior, explora: ↑nodolista
  encontrado: logico
inicio
  obtener_mem (aux)
  aux↑.inf = palabra
  aux↑,.sig = nil
  anterior = nil
  encontrado = falso
  explora = l
  mientras (encontrado = falso) y (explora <> nil)
      Si explora↑.inf < aux↑.inf
          entonces
            anterior = explora
            explora = explora↑.sig
          en otro caso
             encontrado = verdadero
  fin mientras
  aux↑.sig = explora
  Si anterior = nil
     entonces
       l = aux
     en otro caso
       anterior↑.sig = aux
fin
```

lista gracias a que los nodos contienen el campo **sig** para indicar el siguiente nodo de la lista.

El procedimiento inserción dado un puntero **L**, que apunta al primer nodo de la lista, inserta un nuevo nodo con el contenido de la variable **palabra** como información. El procedimiento comienza solicitando memoria para un nuevo nodo, introduciendo la información y procediendo a inicializar los punteros **explora** y **anterior**. La función de estos punteros es la mas importante de este procedimiento. El puntero **explora** se dedica a recorrer la lista en busca del primer nodo cuya información supere a la palabra que debe ser insertada. Cuando se encuentre, **explora** apuntará al nodo al que deberá apuntar el nuevo nodo que se va a insertar. La inserción se produce tras finalizar el bucle **mientras**, que realiza la búsqueda del lugar donde debe ser insertado el nuevo nodo, haciendo que el campo **sig** del nodo al que apunta el puntero **aux** contenga la misma dirección que el puntero **explora**. La última instrucción **si**, está comprobando si se está tratando de insertar el primer nodo de la lista, si esto es así, el puntero **L** debe apuntar a la misma dirección que **aux**, es decir, la lista contendrá ahora un único nodo. Si, por el contrario, la lista ya contenía algún nodo habrá que hacer que el siguiente nodo al que apunta el puntero **anterior** sea el nodo al que está apuntando actualmente el puntero **aux**.

El procedimiento eliminar tiene como parámetros el puntero que apunta al inicio de la lista y la palabra que debe coincidir con la información que contiene el nodo a eliminar. Las variables locales **anterior** y **explora** realizan la misma función que en el procedi-

```
funcion buscar (l: ↑nodolista; palabra: cadena): logico
 explora: ↑nodolista
 encontrado: logico
inicio
 explora = l
 encontrado = falso
 mientras (explora <> nil) y (encontrado = falso)
    Si explora↑.inf = palabra
      entonces
        encontrado = verdadero
      en otro caso
        explora = explora↑.sig
 fin mientras
 Si encontrado = verdadero
   entonces
     buscar = verdadero
   en otro caso
     buscar = falso
fin
```

```
procedimiento eliminar ( var l: ↑nodolista; palabra: cadena)
 anterior, explora: ↑nodolista
 encontrado: logico
inicio
 anterior = nil
 encontrado = falso
 explora = l
 mientras (encontrado = falso) y (explora <> nil)
    Si explora↑.inf = palabra
      entonces
        encontrado = verdadero
      en otro caso
        anterior = explora
        explora = explora↑.sig
    fin si
 fin mientras
 Si encontrado = verdadero
   entonces
     Si anterior = nil
        entonces
          l = explora↑.sig
        en otro caso
          anterior↑.sig = explora↑.sig
     fin si
     Liberar_mem(explora)
 fin si
fin
```

miento de inserción. Si el nodo a eliminar es encontrado, el puntero **explora** apuntará a él, mientras que el puntero **anterior** apuntará al nodo que le precede. Una vez encontrado el nodo a eliminar, el campo **sig** del nodo apuntado por el puntero **anterior** deberá apuntar al nodo al que apunta el campo **sig** del nodo apuntado por el puntero **explora**. Es importante, darse cuenta que el nodo eliminado no es ya necesario, por tanto se utilizará la función **liberar_mem** para disponer de la memoria ocupada por este nodo. Esta es una de las características mas importantes de las estructuras dinámicas de datos, el poder ajustarse a las necesidades de memoria durante la ejecución de un programa.

Por último, indicar que a partir del conjunto de procedimientos y funciones de los cuadros 3, 4, 5 y 6 es posible, combinándolos adecuadamente, realizar operaciones más complejas con esta estructura de datos. Además, se puede pensar en la ineficacia de tratar de mantener ordenada una estructura de datos estática cuando los elementos se insertan en ella ordenándose según van llegando, debido a que hay que desplazar una serie de elementos para dejar hueco al nuevo elemento a insertar.

Puede ser interesante en ocasiones, poder acceder a cualquier nodo partiendo de cualquier punto de la lista. Obsérvese que si se desean realizar dos búsquedas consecutivas se debe partir del primer nodo de la lista, aún cuando los nodos sean consecutivos. Es posible disponer de listas doblemente enlazadas, se diferencian de las anteriores listas en que sus nodos disponen de dos punteros, uno de ellos señalará como hasta ahora al siguiente nodo y el otro puntero apuntará al nodo que le precede. La gestión de estas listas es muy similar al de una lista con un solo enlace.

Es posible, de forma similar a como fueron implementadas las listas ordenadas, construir otros tipos de estructuras usuales dentro de la programación como son las pilas y las colas. No se detallarán en este artículo su implementación completa pero se darán algunas sugerencias para construirlas.
Una pila no es más que una estructura que permite la inserción de elementos al inicio de la estructura y permite recuperar y eliminar la información solamente de su nodo inicial. Así pues, para su implementación bastará con mantener un puntero que indique el principio de la estructura, insertando y eliminando los nodos en dicho punto.

Una cola es una estructura lineal de datos que permite la inserción de sus elementos al final de la estructura y la consulta y eliminación en el inicio de su estructura. Para su implementación, basta con disponer de dos punteros que señalen su nodo inicial y su nodo final, estando todos sus nodos enlazados, como si de una lista enlazada se tratara. Cualquier operación con esta estructura pasará por la modificación de estos dos nodos.
Como se ha visto las posibilidades de implementar mediante memoria dinámica diferentes estructuras es casi infinito.

Simulación de memoria dinámica

Hay lenguajes de programación que no permiten la utilización de memoria

```
PROCEDIMIENTOS PARA LA GESTIÓN DE MEMORIA
nodo : registro
          inf: cadena
          sig: entero
memoria: array (100) nodo

libres : entero
procedimiento inicia_mem
 indice: entero
inicio
 Desde indice = 1 hasta 99
      memoria (indice).sig = indice + 1
 fin desde
 memoria (100).sig = 0
fin
procedimiento obtener_mem (var pto: entero)
inicio
 pto = libres
 libres = memoria (libres).sig
fin

procedimiento libera_mem (pto: entero)
 inicio
 memoria(pto).sig = libres
 libres = pto
 fin
```

```
funcion lista_vacia (l: entero): logico
 inicio
  Si l = 0
    entonces
      lista_vacia = verdadero
    en otro caso
      lista_vacia = falso
 fin

procedimiento listar (l: entero)
  aux: entero
 inicio
  aux = 1
  mientras aux <> 0
   escribir (memoria(aux).inf)
   aux = memoria(aux). sig
  fin mientras
 fin
```

```
procedimiento insertar ( var l:entero; palabra: cadena)
  aux, anterior, explora: entero
  encontrado: logico
inicio
  obtener_mem (aux)
  memoria(aux).inf = palabra
  memoria(aux),.sig = 0
  anterior = 0
  encontrado = falso
  explora = 1
  mientras (encontrado = falso) y (explora <> 0)
     Si memoria(explora).inf < memoria(aux).inf
        entonces
          anterior = explora
          explora = memoria(explora).sig
        en otro caso
          encontrado = verdadero
  fin mientras
  memoria(aux).sig = explora
  Si anterior = 0
    entonces
     l = aux
    en otro caso
      memoria(anterior).sig = aux
 fin
```

dinámica. Esto no quiere decir que no se puedan implementar las estructuras que se han descrito hasta ahora.

Es posible simular memoria dinámica a través de un array implementando adecuadamente las rutinas para obtener y liberar la memoria, además de representar el valor nil.

En los cuadros 7, 8, 9 y 10 se presenta la implementación de una lista enlazada ordenada similar a la utilizada en el anterior apartado utilizando un array como nodos y variables de tipo entero como punteros.

Los diferentes nodos que pueden ser utilizados por la estructura se representarán mediante el array **memoria**. Este array está formado por registros con dos campos, **inf** que guardará la información a tratar y **sig** de tipo entero, que contendrá la posición del array que se corresponde al siguiente elemento de la lista.

Ahora bien, los punteros no son direcciones de memoria, se trata de posiciones del array. Para gestionar adecuadamente los nodos representados por cada una de las componentes del array habrá que inicializar los nodos que no estén ocupados, esta función la realiza el procedimiento **inicia_mem**. Este procedimiento admite una variable entera que apunta al primer elemento del array, consecutivamente los elementos del array guardarán en su campo **sig** la siguiente posición de memoria. Teniendo en cuenta esto, cuando se solicite un nodo, se inspeccionará la lista de nodos libres seleccionando el primero de ellos. La variable que apunta al primer nodo de la lista de nodos libres proporcionará el nodo solicitado a través del procedimiento **obtener_mem** que ahora debe ser implementado por el programador, al igual que el procedimiento **liberar_mem** que permite volver a dejar una componente del array que ya no se necesita.

Una variable de tipo puntero almacena la dirección de memoria donde se pueden encontrar los datos

El valor **nil** estará representado por el valor 0. Cuando el campo **inf** de una componente del array contiene este valor quiere decir que no tiene un sucesor, este valor ha sido seleccionado debido a que el array no posee la componente 0, cualquier otro valor fuera del rango de las componentes del array hubiera servido para este propósito.

Ahora, una lista estará representada mediante una variable entera que apunte a la primera componente del array que forme parte de la lista enlazada. Se podrá acceder a la siguiente componente de la lista a través del campo **sig** que contendrá un valor perteneciente a una componente del array o bien el valor cero en caso de que el nodo no tenga sucesor, esto puede verse claramente en el procedimiento **listar**. En este procedimiento, se comprueba que la lista no está vacía cada vez que se avanza en el recorrido de la lista enlazada, si esto ocurre es posible escribir la información del siguiente nodo que pertenece a la lista y pasar al siguiente nodo que se encuentra anotado en el campo **sig** del nodo actual.

La comprobación de si una lista está vacía o no es trivial, basta con comprobar que la variable entera que representa a la lista contenga un valor entero distinto de cero.

Los procedimientos de inserción, eliminación y búsqueda siguen siendo similares a los procedimientos correspondientes a las listas que se han implementado con memoria dinámica, necesitando en el caso de la inserción y eliminación de las dos variables auxiliares **explora** y **anterior** que en el caso de este tipo de implementación son sendas variables enteras.

Las prestaciones de este tipo de implementación son casi idénticas a la implementación mediante memoria dinámica. El programador será el responsable de las rutinas para la gestión de memoria y evidentemente se utilizará más memoria de la que se necesitará, pero si la estructura es necesaria merece la pena realizar su implementación de una forma similar a como se ha descrito.

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratados en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

CÓMO SUSCRIBIRSE A

SÓLO PROGRAMADORES

Revista práctica para usuarios de PC

Suscríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11 Horario 9 a 14 y 15:00 a 18:00 h.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....Profesión.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

☐ Domiciliación bancaria.
Señor Director del banco .....
Población .....
Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta
le ahorro número.....
l recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.
omo pago de mi suscripción a la revista SÓLO PROGRAMADORES.

CODIGO CUENTA CLIENTE

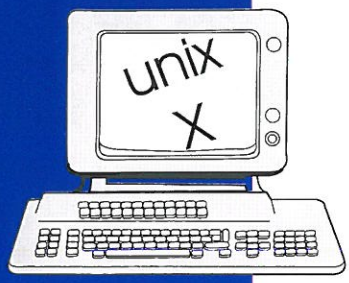
ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28100 Pol. Ind. ALCOBENDAS (Madrid)

DISEÑO DE APLICACIONES DISTRIBUIDAS SOBRE UNIX



Fernando J. Echevarrieta

En el presente artículo se introducen los principios básicos de uno de los mecanismos de distribución de aplicaciones más usados sobre UNIX y disponible en Linux.

Información es poder. Pero sólo es poder si se puede traficar con ella. Hoy en día, todo tiende a la distribución: desde las bases de datos, pasando por cualquier tipo de aplicaciones hasta los mismos sistemas operativos. El presente artículo introduce una de las tecnologías más utilizadas en el desarrollo de aplicaciones distribuidas: el mecanismo de RPC (*Remote Procedure Call*) de SUN. Para ello y con el objeto de que el lector sea capaz de desarrollar sus propias aplicaciones distribuidas se presenta en primer lugar una serie de fundamentos teóricos que, si bien pueden resultar algo difusos, son necesarios para su aplicación posterior al desarrollo de una aplicación ejemplo, basada en RPCs, que se presentará al final del artículo y se desarrollará en el próximo número. Esta aplicación, a la vez de guía y ejercicio para futuros desarrollos, servirá como ejemplo para la asimilación de los conceptos ahora presentados. Para la realización de todo lo expuesto en Linux será imprescindible la instalación del *paquete tcpip* presente en la *serie de discos N*.

PROCEDIMIENTOS REMOTOS

Los procedimientos remotos constituyen una extensión de la idea de procedimiento local. Consisten, como su nombre indica en la llamada y ejecución de un procedimiento que se encuentra en otra máquina. En la mayoría de los casos, esta llamada es bloqueante, es decir, el programa principal detiene su ejecución hasta que el procedimiento retorna, comportamiento análogo al de un procedimiento local.

TRANSPARENCIA EN RPC

La utilización de RPCs no es del todo transparente. Así, es imposible pasar parámetros por referencia ya que ello implicaría la utilización de un sistema único de punteros y descriptores y cada máquina en una red posee su propio espacio de direccionamiento de memoria, aunque ya se esta trabajando en S.O. distribuidos, como el MACH, con espacios de direccionamiento global. Por esta razón tampoco será posible el paso de procedimientos como parámetros.

Otro problema surge de la representación de los datos, que no se puede asumir única en sistemas heterogéneos en los que, por poner el ejemplo más sencillo, se pueden encontrar unas máquinas con representación *big endian* y otras con *little endian*. Por ello, habrá que emplear un *nivel de presentación* o convenio de representación única de la información, que podrá estar basado en un formato externo único, formatos externos múltiples etiquetados o formatos negociados según conexión. Los ejemplos más utilizados son el *XDR* o *eXternal Data Representation* empleado en Internet (mundo TCP/IP) y el *ASN.1* o *Abstract Syntax Notation* version 1 (mundo ISO/OSI).

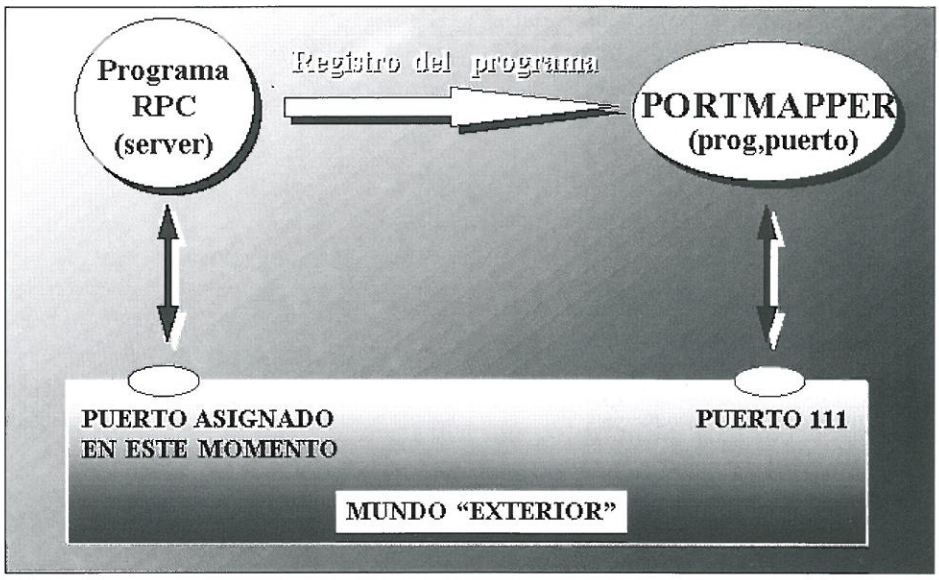
La semántica de las llamadas también cambiará. Ahora, al realizarse la comunicación a través de una red se plantea el problema de que la llamada puede no llegar a destino (no ejecución)

o llegar repetida (múltiple ejecución).

La existencia de la red también plantea problemas de seguridad por lo que será necesario contar con mecanismos de autenticación.

El tratamiento de excepciones es también diferente del de los procedimientos locales siendo ahora la probabilidad de error mayor al presentarse situaciones de servicios no disponibles, servidores que no responden, accesos no autorizados, servidores huérfanos (cuyo cliente ha muerto), etc.

A éstas, habrá que añadir otras cuestiones como el caudal y latencia de la red, cuestiones que en su mayoría no se tratarán en este artículo en aras de la brevedad.



Funcionamiento del portmapper.

SEMÁNTICA DE RPC

Desde el punto de vista semántico, la utilización de un procedimiento remoto no es totalmente transparente al programador debido a que los RPC incorporan una semántica distinta a la de los procedimientos locales. Las semánticas de procedimientos remotos se pueden clasificar en:

*1. **Semántica 1 ó más:*** el llamante insiste en la petición hasta que recibe una confirmación. Con ello queda garantizado que la operación se ha realizado. Sin embargo, debido a que la confirmación puede perderse, es posible que la operación se haya realizado varias veces.

*2. **Semántica a lo sumo 1:*** no se permite que una operación se realice más de una vez. Por ello es necesario el empleo de contadores para el filtrado de duplicados. Sin embargo, una operación puede quedar incompleta.

*3. **Semántica 0 ó 1:*** Obliga a deshacer la operación si esta falla. De este modo, si retorna se tiene la seguridad de se ha realizado exactamente una vez. Si no retorna se tiene la seguridad de que o se ha realizado exactamente una vez o no se ha realizado.

La recuperación de errores se realiza mediante el empleo de plazos y retransmisiones según la semántica empleada. La librería de SUN RPC incluye una serie de timeouts fijos y retransmisiones y no garantiza la *fiabilidad*. El concepto de *fiabilidad* en arquitecturas de protocolos puede consultarse en "Arquitecturas Cliente-Servidor", Sólo Programadores n.17, pág. 23.

TRANSPORTE DE RPC

Las llamadas a procedimientos remotos se montan sobre un *nivel de transporte* y en algunos sistemas se puede elegir el transporte adecuado. Así, en SUN RPC es posible emplear RPC's sobre *UDP* (datagramas) o *TCP* (conexiones). La decisión sobre que nivel adoptar dependerá de la semántica. Así, en UDP, no fiable, existen posibles pérdidas y repeticiones y también hay una limitación del tamaño máximo de los datagramas. El TCP, fiable, implica un mayor esfuerzo de cálculo y latencia pero no presenta límites de tamaño y los RPC no verán duplicados ni pérdidas.

En la mayoría de los casos se sustentan sobre protocolos de datagramas debido a que la utilización de protocolos orientados a conexión supone una mayor complejidad. Sin embargo en algunos casos, es mejor la utilización de conexiones, por ejemplo en la transmisión de ficheros en la que si el parámetro no es el fichero entero, será necesario emplear varios RPCs y varios *ACKs* (confirmaciones) lo que implica un retardo alto para un proceso que podría mejorarse mediante un protocolo de ventana orientado a conexión.

IDENTIFICACIÓN DE LOS RPC

Cada procedimiento remoto queda identificado por un par *(prog, proc)* dónde *prog* es la identificación del programa remoto que viene dada por una dirección de 32 bits y *proc* es la identificación del procedimiento dentro de ese programa dada por un entero. Por convenio se asigna el entero 0 a un procedimiento de test sin funcionalidad (que sirve para comprobar la disponibilidad del programa remoto) y el resto, a los procedimientos útiles del mismo.

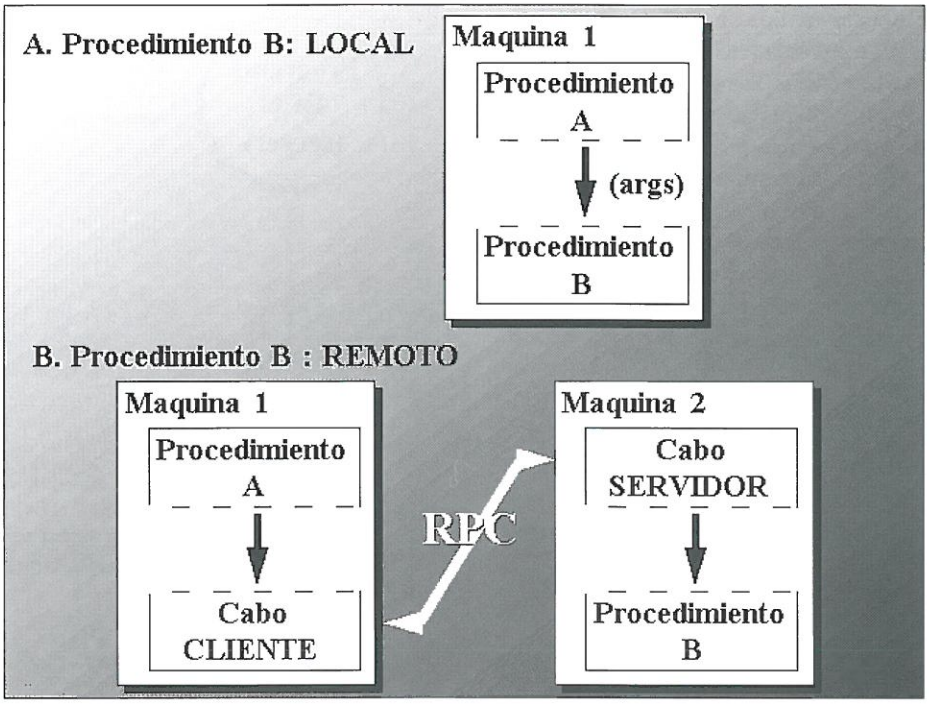
Para permitir la migración de una versión a otra de programas conteniendo procedimientos remotos, así como la realización de pruebas, se permite que coexistan varias versiones de un mismo programa, por lo que, en la práctica, los procedimientos se identifican por una terna *(prog, vers, proc)*, donde *vers* especifica la versión.

Una de las ventajas fundamentales del mecanismo de RPC de SUN es que el propio mecanismo garantiza la corrección del acceso concurrente a datos compartidos, liberando de la gestión de la exclusión mutua al programador. De este modo, queda garantizado que únicamente un procedimiento remoto por cada programa podrá ser ejecutado al mismo tiempo.

PROGRAMAS REMOTOS Y PUERTOS: PORTMAPPER

El nivel de transporte se encarga de proporcionar un mecanismo de transporte de información entre dos puntos finales de comunicación. A estos, se conectarán las aplicaciones cliente y

servidor, quedando para ellas oculta la estructura de la red subyacente. Así pues, bastará conocer la dirección de estos puntos de comunicación para hacer el diálogo posible. En la arquitectura TCP/IP de Internet, una dirección de transporte (nivel 4) viene dada por una dirección IP de nivel 3 (máquina) y un puerto UDP o TCP (punto final de conexión sobre uno u otro protocolo) especificado por un número de 16 bits. De esta forma, cada servicio atiende en un número único de puerto que suele figurar en una lista pública de asignaciones. Pero surge un problema a la hora de asignar un puerto a un programa remoto ya que, como se ha visto, estos quedan identificados mediante números de 32 bits que no caben en los 16 correspondientes a la dirección de un puerto. Así pues, y dado que es improbable que una máquina proporcione a la vez 2 elevado a 16 programas con RPC's se suelen realizar asignaciones temporales dinámicas de los puertos de la máquina a los programas remotos que en ella residen. De esta forma, para utilizar RPCs es necesario haber lanzado en la máquina servidora el programa *portmapper* encargado de realizar estas asignaciones. Este programa, al ser lanzado, siempre se pone a escuchar en el puerto 111. Cuando se lanza un servidor de RPCs llama al puerto 111 para registrarse, proporciona su número de programa fijo de 32 bits y el portmapper le asigna una dirección de puerto de 16 bits. Cuando una máquina cliente desea realizar una llamada remota a uno de los procedimientos del programa servidor, se pone en contacto con el portmapper y le pregunta cuál es en ese momento el puerto correspondiente a la dirección de 32 bits que conoce. El portmapper le facilita el puerto y a él se dirige el cliente para realizar la llamada remota. Este proceso se ha ilustrado en la figura 1.



Distribución de una aplicación con RPCs.

DISTRIBUCIÓN DE PROCEDIMIENTOS REMOTOS

Aunque para proporcionar una base más sólida se han presentado algunos de los conceptos básicos relativos a RPC's, será posible realizar aplicaciones cliente-servidor basadas en este mecanismo sin necesidad de tener amplios conocimientos sobre protocolos ni comunicaciones de red. Por otra parte, aunque el mecanismo SUN RPC se fundamenta en sockets, no será necesario saber nada sobre ellos.

Una llamada es idempotente cuando su efecto es el mismo se utilice una o más veces

El diseño de aplicaciones distribuidas puede realizarse orientado a la comunicación, comenzando por el diseño del protocolo (formato de mensajes y sintaxis) u orientado a la aplicación (realización de la misma y posterior división en piezas). El mecanismo RPC corresponde a este último paradigma en el que los límites de las piezas corresponden a las fronteras entre procedimientos.

Así pues, en el nivel a estudiar, el programador parte de un programa de aplicación local en el que las interfaces con los procedimientos están constituidas por su nombre y los argumentos que se les pasa como muestra la figura 2a. Si ahora se desea que el procedimiento A resida en una máquina cliente y el B en una servidora, será necesario añadir nuevo código. En el lado del cliente habrá que incluir código de *serialización de argumentos* (traducción a un formato de representación independiente de la arquitectura, como XDR), generación del mensaje de llamada a procedimiento remoto, envío del mismo y espera y decodificación de resultados del formato externo al interno. Análogamente, en el caso del servidor, será necesario añadir código de servidor que acepte peticiones de RPC, decodifique los argumentos del formato externo al interno, los entregue al procedimiento adecuado dentro del programa remoto, serialice la respuesta, genere el mensaje de respuesta RPC y lo envíe. Afortunadamente, entre las herramientas proporcionadas por SUN se encuentra el programa *rpcgen* que realizará la mayor parte de este trabajo de manera automática y se estudiará en el próximo artículo.

Como en todo momento se desea que la aplicación original cambie en lo mínimo posible y el mecanismo sea lo más transparente posible al programa-

dor, el medio más elegante es la inclusión de este código adicional en dos *procedimientos stub* (terminología anglosajona que se encontrará en la documentación en línea y en la mayor parte de la bibliografía, desgraciadamente, en inglés) o, en castellano, *cabos*, uno en el lado del cliente y otro en el lado del servidor, por cada procedimiento remoto. De esta forma, no es necesario modificar los procedimientos del programa original. Así, como muestra la figura 2b, el cabo del cliente presentará el mismo interfaz que el procedimiento B y se encargará de todo el proceso de comunicación, serialización, etc junto con el cabo del servidor y el interfaz de este con el procedimiento B será el mismo que presentaba el A, es decir, invocará a B y recibirá sus resultados como si fuera el propio procedimiento A, encargándose posteriormente de devolverlos al cabo del cliente. De esta forma, los procedimientos A y B del programa original permanecerán inalterados. El mecanismo puede incluso permitir una transparencia de interfaz total si se nombra al cabo cliente con el mismo nombre que al procedimiento B (no habrá conflictos porque el procedimiento B ahora reside en otra máquina y aunque estuviera en la misma formaría parte de otro módulo de biblioteca), lo que permitirá incluso mantener intactos los códigos objeto de los procedimientos originales.

PAQUETES DE USUARIO

Como se ha visto, la diferencia en la semántica, es uno de los principales problemas de transparencia en programación con procedimientos remotos. Por ello, en la mayoría de los casos se suelen desarrollar *paquetes de usuario* para el cliente, que no son más que conjuntos de procedimientos que proporcionan al programador una interfaz con la semántica deseada e internamente hacen uso de las funciones proporcionadas por el cabo del cliente que, as su vez, realizará las llamadas a RPC.

Para ello se estudiará el caso de semántica 1 ó más, lo que obligará a la utilización de *llamadas idempotentes*. Una llamada es idempotente cuando su efecto es el mismo se utilice una o más veces. Así, la definición de procedimientos como *read(fichero,numbytes)* no será correcta, ya que al realizar una lectura secuencial proporcionará resultados distintos en cada ejecución. Una definición correcta de un RPC idempotente para el caso, será *read(fichero,numbytes,offset)*, que siempre dará el mismo resultado.

De esta forma, un paquete de usuario sólo llamará a procedimientos idempotentes, por lo que deberá almacenar información de estado. Esto, como se podrá observar en el ejemplo siguiente, a veces supone también una optimización del acceso a la red.

EJEMPLO DE INTERFAZ DE UN PAQUETE DE USUARIO

Supóngase un sistema de acceso remoto a ficheros sustentado sobre RPC que proporcione una interfaz idéntica a la API del sistema de ficheros. La idea no es, en absoluto descabellada ya que, por ejemplo, el *NFS* (*Network File System*) de SUN cumple estas características. Debido a que esta interfaz no consta de llamadas idempotentes, se desarrolla un paquete de usuario que realice la conversión de semántica y almacene información de estado.

El paquete podría presenta la siguiente interfaz de funciones:

open(fichero)

Abre el fichero remoto indicado y devuelve un descriptor.

read(descriptor, buffer, numbytes)

Lee *numbytes* del fichero indicado por el descriptor y los almacena en buffer. Devuelve el número de bytes leídos. La siguiente lectura se realiza donde terminó la anterior.

close(fichero)

Cierra el fichero indicado.

Como se puede observar, estas llamadas (idénticas a las del S.O.) no son idempotentes, como se explicaba en el apartado anterior. El paquete de usuario deberá, por tanto, realizar una conversión de semántica para lo que utilizará funciones idempotentes (figura 3) del cabo cliente, que se asentarán sobre el mecanismo de RPC's. Por ejemplo:

busca(fichero)

Comprueba si el el fichero indicado existe.

read_idemp(descriptor,offset,SIZE)

Lee con acceso aleatorio SIZE bytes a partir del offset indicado en el fichero

El portmapper es el programa encargado de asignar los puertos a los programas remotos

LISTADO 1

```
/* credit.c = Aplicacion local con dos procedimientos */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

#define MAXLEN 80

char fich_claves[MAXLEN]="fichero";

main()
{
        char nombre[30];
        int  numero;

        pidedatos(nombre,&numero);
        if (confirmadatos(nombre,numero) == 1)
          printf("CONFIRMADO\n");
    else
          printf("DENEGADO\n");
}

pidedatos(char *nombre, int *numero)
/* Procedimiento A */
{
        printf("Nombre: "); scanf ("%s",nombre);
        printf("Numero: "); scanf ("%d",numero);
}

confirmadatos(char *nombre, int numero)
/* Procedimiento B */
{
        char linea[MAXLEN], numasc[20];
        FILE *fichero;
        int retorno=0;

        sprintf(numasc,"%d",numero);

        fichero=fopen(fich_claves,"r");

        while (fgets(linea,MAXLEN,fichero)
        != NULL)
            if ((strcmp(strtok(linea,","),numasc)
            == 0 ) &&
              (strcmp(strtok(NULL ,","),nombre)
              == 0 ) &&
              (strcmp(strtok(NULL ,"""),"OK\n")
              == 0 ))
              retorno=1;

        close(fich_claves);
        return retorno;
}
```

Aplicación local de partida.

LISTADO 2

```
1,echeva,OK
2,eva,OK
3,juan,CANCELADA
4,echeva,OK
5,echeva,CANCELADA
```

Ejemplo de fichero gestionado por la aplicación.

especificado por el descriptor y los introduce en un buffer interno asociado. SIZE es una constante definida en el programa.

Cuando se llama a la función *open* que proporciona el paquete de usuario, éste ejecuta la función *busca* sustentada sobre RPCs. El programa remoto únicamente confirmará la existencia del fichero y los permisos de acceso indicados por el *open*, sin abrirlo.

Cuando se llama a la función *read* que proporciona el paquete de usuario, éste ejecuta la función *read_idemp* sustentada sobre RPCs cuidando de guardar el estado del offset y del número de bytes realmente leídos y ocultándolo al programa de aplicación. El programa remoto realizará un *open*, *lseek*, *read* (acceso directo) y *close* del fichero.

EJEMPLO DE FUNCIONAMIENTO DE UN PAQUETE DE USUARIO

Imagínese un programa de aplicación que haciendo uso de las funciones del paquete de usuario contuviera el siguiente código:

[1] *fd=open("Fichero.txt");*
[2] *n=read(fd,buffer,712);*
[3] *n=read(fd,buffer,312);*
[4] *n=read(fd,buffer,100);*
[5] *close(fd);*

Esto produciría las siguientes acciones del paquete de usuario y el programa remoto (suponiendo SIZE=1024):

[1] *busca("Fichero.txt")*
Si el fichero existe, el paquete genera un descriptor interno suyo para referirse al mismo y lo devolverá como valor de retorno. El programa remoto ejecutará una llamada *access*, pero no un *open*.
[2] *lee(desc,0,1024);*
El programa remoto ejecutará un *open*, *lseek*, *read* y *close* del fichero. La lectura será de 1K aunque sólo se haya solicitado 712 bytes. Pero el programa de aplicación sólo recibirá lo que solicitó.
[3] *No producirá ninguna llamada a RPC.*
La cantidad solicitada aún se encuentra en el buffer interno del paquete de usuario por lo que éste devolverá los datos sin necesidad de generar RPCs.
[4] *lee(desc,1024,1024);*
Es necesario hacer una nueva lectura. Como la llamada es idempotente, el paquete de usuario debe recordar el offset de la llamada anterior. Nuevamente se lee 1k aunque se devuelven sólo 100 bytes. Otra vez, el programa remoto realizará las cuatro acciones sobre el fichero remoto: *open*, *lseek*, *read* y *close*.

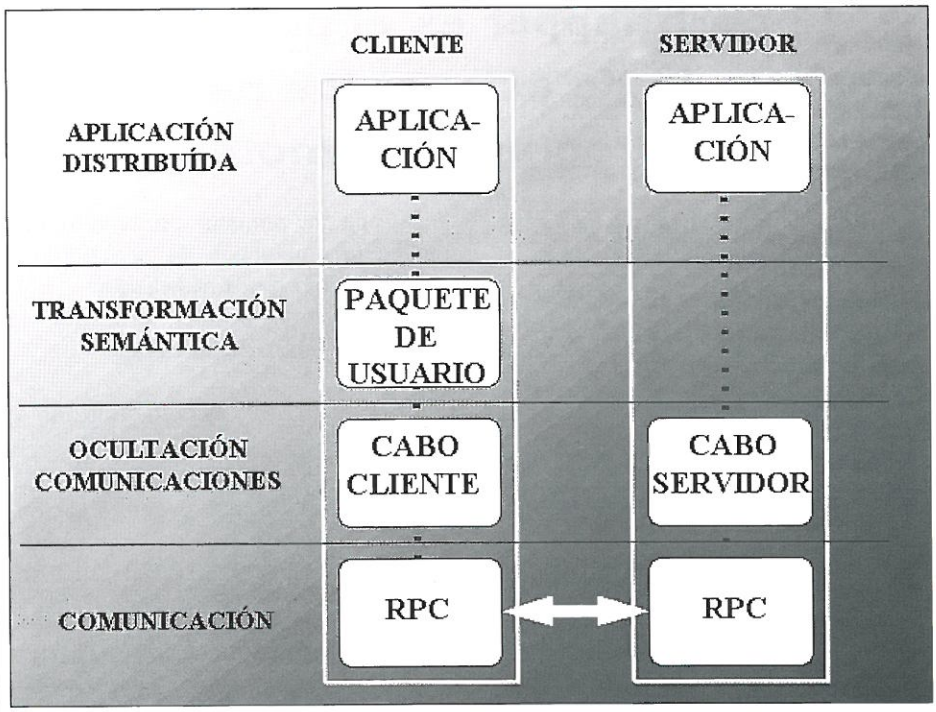
De esta forma, se ha logrado hacer transparente el uso de RPCs. El programador de la aplicación hará uso de las funciones facilitadas por el paquete de usuario sin necesidad de conocimientos previos sobre comunicaciones, de la misma forma que emplea las primitivas del sistema operativo. Del mismo modo, se observa como el estado almacenado por el paquete de usuario puede optimizar el acceso a red, por ejemplo, en el caso [3], en el que no se ha realizado ningún acceso.

LISTADO A

```
/* credit1.c = Aplicacion local con el procedimiento
A */
/* echeva@dit.upm.es para Solo Programadores
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

main()
{
        char nombre[30];
        int  numero;

        pidedatos(nombre,&numero);
        if (confirmadatos(nombre,numero) == 1)
          printf("CONFIRMADO\n");
      else
          printf("DENEGADO\n");
}

pidedatos(char *nombre, int *numero)
/* Procedimiento A */
{
        printf("Nombre: "); scanf ("%s",nombre);
        printf("Numero: "); scanf ("%d",numero);
}
```

Aplicación local con el procedimiento A.

Así pues, se pueden distinguir diferentes niveles para el mantenimiento de la transparencia: el cabo cliente mantendrá el código adicional de comunicaciones oculto a la aplicación y el paquete de usuario realizará la conversión de semántica. En cualquier caso, hay que resaltar que el paquete de usuario puede no ser necesario,



Funcionamiento de un paquete de usuario.

LISTADO B

```
/* credit2.c = Aplicacion remota con el procedimien-
to B */
/* echeva@dit.upm.es para Solo Programadores
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

#define MAXLEN 80

char fich_claves[MAXLEN]="fichero";

confirmadatos(char *nombre, int numero)
/* Procedimiento B: remoto */
{
        char linea[MAXLEN], numasc[20];
        FILE *fichero;
        int retorno=0;

        sprintf(numasc,"%d",numero);

        fichero=fopen(fich_claves,"r");

        while (fgets(linea,MAXLEN,fichero)
!= NULL)
            if ((strcmp(strtok(linea,","),numasc)
== 0 ) &&
                (strcmp(strtok(NULL ,","),nombre)
== 0 ) &&
                (strcmp(strtok(NULL ,""),"OK\n")
== 0 ))
                retorno=1;

        close(fich_claves);
        return retorno;
}
```

Aplicación remota con el procedimiento B.

como en el caso de la aplicación ejemplo que se va a desarrollar.

GENERACIÓN DE UNA APLICACIÓN DISTRIBUIDA

Una vez expuestos los conceptos necesarios, se va a desarrollar una aplicación distribuida basada en RPC's. Por razones de espacio y con el objeto de no perderse en detalles de programa, la aplicación será casi tan simple como un "hello world"; sin embargo, servirá para ilustrar los pasos fundamentales en la generación de aplicaciones distribuidas con un diseño orientado a la aplicación mediante *rpcgen*. Para ello, se ha diseñado una simple aplicación distribuida como pueda ser la de consulta de la validez de una tarjeta de crédito a una base de datos remota. Así pues, se dispondrá de un programa cliente que bien se podría facilitar como una llamada a un módulo de biblioteca, que tras pedir los datos del cliente realizará una consulta a la base de datos central.

El estado almacenado en un paquete de usuario puede servir también para optimizar el uso de la red

En un ejemplo tan simple, la mayor parte del código será el dedicado a comunicaciones. Sin embargo, éste no crecerá mucho más con aplicaciones complejas.

CONSTRUCCIÓN DEL PROGRAMA LOCAL

Al partir de un diseño orientado a la aplicación, se realizará primeramente un programa local. Esta metodología, si bien siempre es útil para migrar aplicaciones de locales a distribuídas no será necesaria una vez se disponga de más experiencia. El listado 1 presenta la aplicación de partida que, por su sencillez, es autoexplicativa.

DIVISIÓN EN PARTES

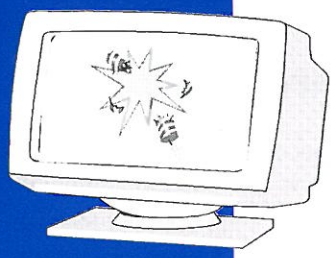
El segundo paso será la división de la aplicación en los fragmentos que serán distribuidos. En el caso del ejemplo, esta división no dará mucho que pensar. Sin embargo, en aplicaciones complejas será necesario contar con un claro esquema conceptual de las llamadas RPC que se producirán para realizar una partición eficiente. Así pues, será necesario conocer los recursos de los que hará uso cada procedimiento para alojarlos en la máquina en que reside el recurso. Así, si el procedimiento A, *pidedatos*, lee de la entrada estándar, deberá residir en la misma máquina que el main. Análogamente, si el procedimiento B, *confirmadatos*, accede a un fichero, será conveniente que se ejecute en la misma máquina en la que se encuentra el fichero. En el ejemplo se tendrá una máquina local con un programa local con el main y *pidedatos* que realizará llamadas al procedimiento remoto *confirmadatos* que formará parte de un programa remoto ejecutándose en la máquina que contenga el fichero con la información relativa a las tarjetas de crédito. Por supuesto, el programador deberá separar las constantes y estructuras de datos utilizadas por cada procedimiento y definirlas en los ficheros correspondientes. Las declaraciones de datos globales deberán figurar en ambos.

Es conveniente alojar los procedimientos en las máquinas en que se encuentran los recursos de los que harán uso

EL PRÓXIMO MES

Al igual que ocurrió con los mecanismos de sockets, este mes se ha realizado una introducción fundamentalmente teórica que será complementada el próximo con la transformación de esta sencilla aplicación en una aplicación distribuida. Aproveche este tiempo para releer detenidamente el artículo ya que serán muchas las variables que haya tocar y es conveniente haber asimilado el máximo número de conceptos posible. La generación de aplicaciones mediante RPCs, no es difícil ya que es una técnica bastante mecánica. Sin embargo, a la vez es compleja.

Mientras tanto, podrá ponerse en contacto con el autor mediante carta a la redacción de la revista o por:
E-mail internet: echeva@dit.upm.es
E-mail Compuserve: 100676,2456
WWW:
http://highland.dit.upm.es:8000



EL EFECTO BLUR MOTION

Pedro Antón

Aunque el efecto Blur Motion consiste en dejar una estela cuando una figura o sprite está en movimiento en la pantalla del ordenador, el efecto que se explica en este artículo, no consiste únicamente en eso. En este artículo, quiero animar a todos los lectores a que experimenten con los efectos que conocen o han ido aprendiendo a lo largo del curso, de esta manera, llegarán a conseguir algo novedoso o cuando menos, conseguirán hacer efectos no idénticos a los comunes. Además debido a la cantidad de dudas con respecto a la integración del sistema de sonido incluido en el número 16 Solo Programadores, se comentará como integrarlo en un programa ensamblador.

color 0 que ha de ser el color del fondo. Es de vital importancia definir la paleta correctamente, esto es, degradarla de forma deseada, partiendo del color más oscuro al más claro con el que se vaya a trabajar, dentro de los márgenes de color que se desee establecer. Si la progresión es de uno, el efecto se consigue, como se indica en el dibujo.
Organigrama del efecto Blur

LA PALETA

Es imprescindible un correcto degradado.

Al igual que ocurría con el fuego, es imprescindible una paleta correcta. Ahora bien, en algunos casos al lector no le interesará disponer de la paleta completa. En ese caso, el lector debe

EL EFECTO BLUR MOTION

Los modos de tratar el buffer de trabajo, dan lugar a efectos como este.

El Blur motion es un efecto realmente sencillo de conseguir, consiste en decrementar los colores en el buffer de forma progresiva, hasta llegar al

tener en cuenta que en una VGA en modo 13h, el máximo número de tonalidades de un solo color RGB que se puede conseguir es 64. Quiere decir esto, que sólo es posible degradar cualquiera de los colores puros: rojo, verde y azul (RedGreenBlue) desde su máxi-

Se denomina Blur Motion al efecto que se consigue cuando una figura o sprite deja una estela a su paso.

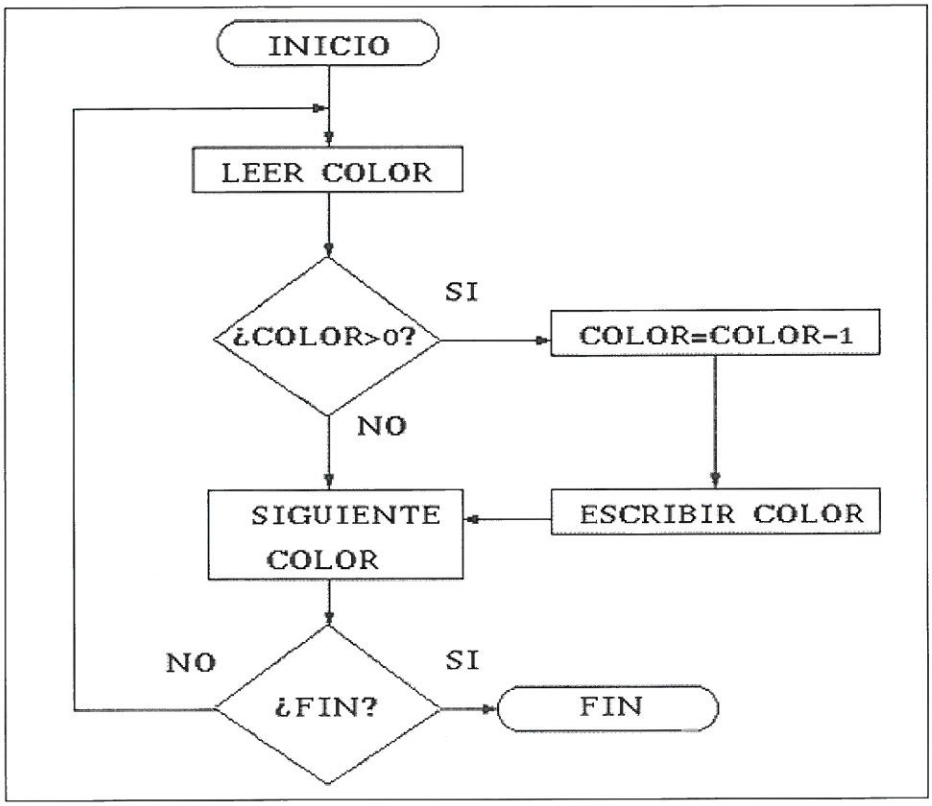


Diagrama de flujo del efecto BlurMotion.

ma luminosidad a la mínima en 64 tonos, siempre y cuando se emplee un DAC de 8 bits. Con esto se intenta explicar al lector, la posibilidad de hacer blur motion a una figura o sprite en movimiento, tratando los 64 primeros colores, sin perder resolución en la distorsión, y a su vez, conservar otra zona de memoria, con otro efecto, o no, sin que se vea afectada por dicha distorsión.

En el código que acompaña a este artículo, se usan dos colores base distintos, no azul puro, empleando la totalidad de la paleta, para conseguir un degradado muy suave.

EL BUFFER

Ni que decir tiene, que todo este proceso se realizará en un buffer, con el fin de evitar, en todo momento, que el retrazo vertical pueda estropear el efecto, por no hablar de la velocidad. Además, no necesariamente, se trabajará con la totalidad del buffer. En el código que acompaña a este artículo, se puede ver como no se trabaja con la zona inferior del buffer, de esta forma el gráfico inferior no es modificado aunque se use la totalidad de la paleta.

MODIFICACIÓN DEL EFECTO

Experimentar a la hora de crear algo, puede resultar sorprendente.

Cuando se creó este código, David Tolosana, autor de la canción que acompaña al efecto, lo bautizó como Blur Fire. El motivo, es simple, se consigue el, cuando menos curioso, difuminado, aplicando al efecto, los mismos cálculos que se emplearon para crear el fuego.

Esto es, calcular la media aritmética del valor del punto y otros tres adyacentes a este. Como el lector recordará, se obtendrán distintos resultados dependiendo de los puntos que se traten y del comienzo del tratamiento. Quiere esto decir, que no es lo mismo empezar a distorsionar el buffer por el principio que por el final, y que no es lo mismo, tomar el punto superior al punto que se está tratando que el inferior.

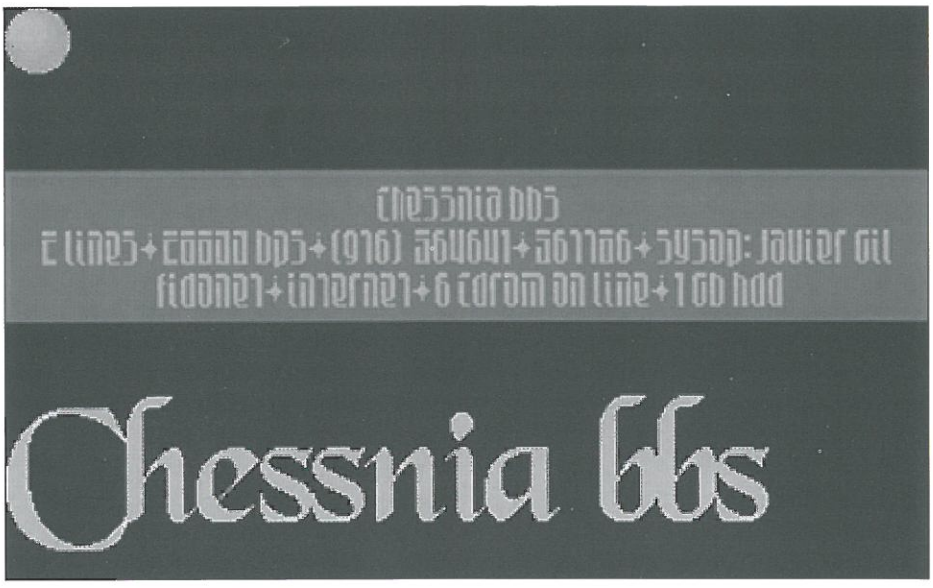
Aplicar los cálculos que se emplean para crear el fuego, da lugar a este efecto. Con esto quiero animar a los lectores a que experimenten con los efectos que vayan consiguiendo, en más de un caso os sorprenderéis del resultado.

UN PROGRAMA ENSAMBLADOR

La creación de un programa en ensamblador es sencilla.

Para aquellos lectores que nunca hayan creado un programa integro en ensamblador, se comentará brevemente la estructura que este debe tener:

La directiva TITLE es algo meramente ilustrativo.



Las directivas DOSSEG y MODEL indica al compilador que debe usar el orden y el nombre de los segmentos de Microsoft.

La directiva .386 indica al compilador que el programa usará, código 386, es decir, registros e instruccio-

Tratamiento de las interrupciones

```
;+++++++++++++++++++++++++++++++++++++++
; Here comes the timer interrupt
;              Call    OldInt8    =    PLAYER
;+++++++++++++++++++++++++++++++++++++++
MyInt8  PROC
        INC   CS:CntStep
        JMP   DWORD PTR [CS:OldInt8]
        IRET
MyInt8  ENDP

;+++++++++++++++++++++++++++++++++++++++
; I N T   9   we don't want PAUSE key :) ONLY
ESC
;+++++++++++++++++++++++++++++++++++++++
MyInt9  PROC
        PUSH  AX
        IN    AL,60h
        PUSH  AX
        IN    AL,61h
        MOV   AH,AL
        OR    Al,80h
        OUT   61h,AL
        POP   AX
        SHL   AL,1
        SHR   AL,1
        CMP   AL,1
        JNE   @@NoEsc
        MOV   CS:[EscKey],1
@@NoEsc:
        MOV   AL,20h
        OUT   20h,AL
        POP   AX
        IRET
MyInt9  ENDP
```

nes, propias, únicamente del 386 y microprocesadores superiores de Intel.

.STACK 2048 define 2048 bytes para el "stack" o pila.

A continuación, se definen los segmentos extras que el programa vaya a usar, en nuestro caso, se usará uno para el buffer y otro para los gráficos que posee el programa, estos están transformados de formato RAW a bytes, en forma de DB. De esta forma, usando la directiva INCLUDE se incluyen los gráficos al programa principal, de forma limpia y clara.

Ahora se pueden definir las MACROS. Una macro es un fragmento de código que será incluido tal cual, en la parte del programa donde sea llamado.

A continuación se definen las variables en el segmento de datos. Este segmento comienza con la directiva .DATA, si anteriormente se han usado las directivas DOSSEG y MODEL, como es nuestro caso. Nótese la inclusión de bytes conteniendo datos, como la canción, la tabla de senos, o la paleta, por mediación de la directiva INCLUDE.

Acto seguido se define el segmento de código, donde se incluye la librería OPL2LIB mediante la ya conocida directiva INCLUDE. Esto incluirá toda la parte del sistema de música en el segmento de código, que, en nuestro caso,comienza con la directiva .CODE. El motivo de incluir las variables del "player" en el segmento de código es evidente, el sistema de música se engancha a la interrupción 8, la cual se ejecuta 18.2 veces por segundo, ese tiempo es empleado por el sistema de música para tocar la melodía, y resulta más cómodo tener los datos y el código en un único segmento, esta interrupción se usará, como veremos más adelante, para fijar el número de frames máximo, a 18.

Es el momento de comenzar a programar. Al igual que en los lenguajes de alto nivel, se pueden definir procedimientos, mediante las directivas PROC y ENDP. Resaltar la importancia que tiene la instrucción RET al final de cada uno ellos, de otra forma el procesador se perdería, puesto que el registro IP (Instruction Pointer) continuaría avanzando de forma inadecuada.

Vista esta pequeña introducción sobre la estructura de un programa en ensamblador, se procederá a explicar el programa que acompaña a este artículo.

DESCRIPCION DE LAS VARIABLES

A continuación se describirán las principales variables del código. Son destacables las siguientes variables:

OldInt8 y *OldInt9* son las direcciones de memoria donde se almacenarán los vectores de interrupción de las interrupciones 8 y 9, con el fin de poder restaurar dichos vectores adecuadamente, al finalizar la ejecución del programa.

Las variables *InitRad*, *EndRad*, *IncRad*, *IncAng*, *CntAng* controlan el movimiento de la bola que gira circularmente, representan el radio con el que comienza a girar, el radio al que debe llegar, el sentido que lleva el incremento del radio, es decir, si el radio a de incrementarse o decrementarse, el contador de ángulo, que indicará en la posición en la circunferencia y el incremento que se le va a aplicar al contador de ángulo.

La variable *PlayM* almacena un uno si el controlador de música ha detectado un chip OPL2 o superior y un cero si no lo ha hecho, con la finalidad de poner el logo de forma aleatoria en caso de no detectar chip OPL, ya que en caso contrario, el logo se pondrá a ritmo de la batería que va en el canal 9.

La variable *CntStep* le resultará curiosa al lector, esta variable almacena un valor, que es usado para detener el proceso del programa, en el caso de que la maquina sobre la que se esté ejecutando la aplicación, sea capaz de generar más de 18 frames por segundo.

La variable *EscKey* contendrá un 1 cuando se pulse la tecla de ESC.

Se pueden definir procedimientos mediante los directivos PMOL y ENOP

DESCRIPCION DE LOS PROCEDIMIENTOS

El procedimiento BlurFire es el alma del efecto.

Algunos de los procedimientos que emplea el programa no serán comentados, ya que han sido explicados en anteriores artículos de este curso.

Los procedimientos *MyInt8* y *MyInt9* se encargan de gestionar las interrupciones 8 y 9 de la forma que nosotros creemos conveniente.

El procedimiento *LookEqu*, será de especial interés, para todos aquellos

$$x = \frac{InitRad \cdot cos \cdot (CntAng)}{256}$$

$$y = \frac{InitRad \cdot sen \cdot (CntAng)}{256}$$

lectores que hayan tenido problemas con el sistema de música. En caso de estar sonando la melodía, comprueba el valor de la variable *Equalize* del canal 9, si este es igual a 14, pone el Logo en una posición aleatoria en el buffer. Si no suena la melodía, pone el logo cuando obtiene un valor cero de un numero aleatorio entre 0 y 7.

El procedimiento *PutBall*, calcula las coordenadas (x,y) donde se va a poner el sprite de la bola, los cálculos a realizar para calcular dichas coordenadas son:

La división entre 256 se realiza debido al uso de punto fijo de 8 bits para almacenar los datos en la tabla de senos, esta división se realizará rotando el valor 8 bits a la derecha, como ya se ha visto en otros artículos de este curso.

EL BUCLE PRINCIPAL

A continuación se analizará el cuerpo del programa, como siempre, tendrá una parte de inicialización de variables, el cuerpo, y la salida del programa.

En la inicialización de variables, se coloca DS en la dirección de memoria ocupado por las variables, y se almacena en las variables *SegBuffer* y *SegGraph*, las direcciones de memoria donde se encuentran el buffer y los gráficos respectivamente. Resaltar, también, la captura de los vectores de interrupción de las interrupciones 8 y 9 y el posterior redireccionamiento de dichos vectores a los gestores de interrupción propios del programa. A continuación se inicializa el sistema de sonido, si existe, así como las variables que usa el procedimiento *PutSpriteClip*, explicado en el anterior artículo de este curso.

En estos momentos comienza el bucle principal del programa. Los pasos a realizar son los siguientes:

Comprobar si las variables *InitRad* y *EndRad* tienen idénticos valores, en caso de ser así se calcula de forma aleatoria los valores de las variables *EndRad* e *IncAng*.

A continuación se coloca el sprite con la bola en el buffer.

Actualizamos la variable *InitRad*, incrementándola o decrementándola, según el valor de *IncRad*. Recordemos, que esta variable contiene uno si dicha variable ha de ser incrementada, y cero si ha de ser decrementada.

El siguiente paso, será poner el logo, si corresponde. Como ya se ha explicado anteriormente dependerá de la existencia, o no, del chip de síntesis FM OPL2, o superiores.

Ahora se debe volcar el buffer en la memoria de vídeo, pero se está usando una variable de sincronismo encadenada a la interrupción 8, no se volcará el buffer a la memoria de vídeo hasta que esta variable contenga un valor distinto de cero y posteriormente se produzca un retrazo vertical.

El procedimiento PutBall calcula las cordenadas (x, y) donde se va a poner el sprite

BLUR FIRE

```
;
;  Here comes the timer interrupt
;  Call OldInt8 = PLAYER ;
MyInt8    PROC
          INC     CS:CntStep
          JMP     DWORD PTR [CS:OldInt8]
          IRET
MyInt8    ENDP

;
;  I N T   9   we don't want PAUSE key :) ONLY
ESC
;
MyInt9    PROC
          PUSH    AX
          IN      AL,60h
          PUSH    AX
          IN      AL,61h
          MOV     AH,AL
          OR      Al,80h
          OUT     61h,AL
          POP     AX
          SHL     AL,1
          SHR     AL,1
          CMP     AL,1
          JNE     @@NoEsc
          MOV     CS:[EscKey],1
@@NoEsc:
          MOV     AL,20h
          OUT     20h,AL
          POP     AX
          IRET
MyInt9    ENDP
```

Es necesario actualizar la variable *CntAng*, sumándole el valor que contiene la variable IncAng, este es un valor aleatorio, el cual, es modificado cuando *InitRad* y *EndRad* poseen idénticos valores.

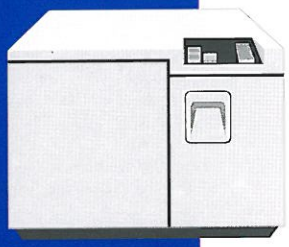
Este proceso se repetirá hasta que la variable *ESCKey* pase a 1. Eso ocurrirá cuando la tecla de ESC sea pulsada, con esta finalidad y la de controlar la tecla de PAUSE, se capturó al principio del programa la interrupción 9.

Del bloque principal, destacar, la parte que espera por el sincronismo, si el ordenador que ejecuta este programa puede hacer más de 18 frames por segundo, cuando llegue a esta parte del código, el ordenador se parará hasta que la variable *CntStep*, contenga un dato distinto de cero, dicho dato sólo es modificado por nuestro gestor de interrupción 8. En el caso contrario, si el ordenador no consigue hacer 18 frames por segundo, cuando el programa llegue a este punto la variable *CntStep*, contendrá un valor superior a uno, en cuyo caso sigue adelante.

Para finalizar con este artículo, nunca se debe olvidar restaurar los vectores de interrupción que se hayan modificado, así como descargar el sistema de sonido, en caso de estar activo.

FICHEROS EN MVS (I)

José María Peco



En cualquier sistema operativo, la información se encuentra agrupada en conjuntos homogéneos, recibiendo cada uno de estos conjuntos la denominación de fichero, archivo o dataset.

En MVS, la etiqueta o nombre de un fichero se forma agrupando calificadores y separando estos con un punto. Cada uno de los calificadores puede tener una longitud máxima de 8 caracteres alfanuméricos y no puede empezar ni por dígito ni por un carácter especial, con excepción del $ que es valido.

La longitud máxima del nombre de un fichero no puede superar los 44 caracteres, y debe tener al menos dos calificadores, exceptuando los ficheros temporales, cuyo nombre se forma anteponiendo un & a un único calificador.

El nombre del miembro de un fichero particionado (tipo de fichero que se describe posteriormente), se forma escribiendo el nombre del fichero, de acuerdo con las reglas descritas anteriormente, y entre paréntesis el nombre del miembro a procesar, que solo podrá tener hasta 8 caracteres.

Otro término contemplado en MVS es el *ALIAS* para los nombres de ficheros, siendo estos ALIAS nombres alternativos dados a ficheros. Para las versiones anteriores al MVS/ESA este nombre alternativo sólo podía especificarse para el primer cualificador. En cambio, con esta versión de MVS, el ALIAS puede referenciar los cuatro primeros calificadores si así se desea.

Estos ALIAS residen en el catalogo maestro, término que también se vera posteriormente, y apuntan a un determinado catalogo de usuario, donde estará ya la ubicación física de un determinado fichero.

DEFINICIONES

Antes de entrar en materia, conviene recordar algunas definiciones de términos comunes en estos temas:

Soporte: Lugar donde se encuentra almacenada la información.

Campo: Unidad elemental de información procesable de un registro.

Es la unidad más pequeña de información que nos puede interesar.

Registro Lógico: Conjunto de campos. Es la unidad básica de información con significado propio para el usuario o informático.

Bloque o Registro físico: Es la unidad de transferencia entre los dispositivos y la memoria, o viceversa.

Factor de bloqueo: Número de registros lógicos contenidos en un registro físico, o bloque.

Fichero o archivo: Conjunto de registros lógicos referidos a un mismo tema

Volumen: Conjunto de información a la que se puede acceder en un instante determinado, en un cierto dispositivo.

Características de los ficheros que influyen en su organización

Tamaño: viene determinado por el número de registros de que consta. Para este concepto debe tenerse en cuenta, no sólo el tamaño actual, sino el previsible.

Volatilidad: es el número de altas y bajas que se procesan en cada pasada del fichero. Por tanto, un fichero que sea muy volátil deberá ser reorganizado muy a menudo.

Actividad: La actividad es la frecuencia con la que se actualiza o referencia un fichero. Así, si un fichero

A nivel de programación, es no sólo importante sino también necesario que se conozcan las herramientas que pone a su disposición el S.O. para manejar la información, que en el entorno de los grandes sistemas puede llegar a ser muy voluminosa

sufre muchas actualizaciones, diremos que es muy activo.

Magnitud de actividad: en un proceso por lotes se denomina magnitud de actividad al número de pasadas por unidad de tiempo.

Porcentaje de actividad: es el porcentaje de registros que se actualizan o referencian en cada pasada. Cuando el numero de transacciones a realizar sea mayor del 10 % no compensa realizar procesos al azar, sino que es mejor hacer un proceso *batch* secuencial.

Distribución de actividad: Es la distribución que se hace de los registros para que tengan mayor posibilidad de acceso, y por lo tanto mayor rapidez para acceder a los registros. La magnitud de actividad y el porcentaje son inversamente proporcionales. Cuanto más pasadas se hagan, será menor el numero de registros referenciados.

Unidad de disco magnético

Así mismo, y también como introducción al tema, conviene recordar la composición y estructura física de las unidades de disco, ya que esta influye muy directamente en la definición del fichero como se verá en los siguientes apartados.

Una unidad de discos es un *armario* en el que se coloca uno o varios *dispack*, normalmente 2; estando cada *dispack* constituido por varios discos concéntricos.

Para leer o grabar información en un disco, se utilizan las cabezas de lectura-grabación. Estas cabezas se encuentran en los extremos de los brazos de acceso, y estos a su vez se encuentran anclados a un peine fijo, tal y como se muestra en la **figura 1**.

Entre dos discos entra un brazo de acceso con dos cabezas (puede existir mas de una cabeza por pista). Todos los brazos de acceso se mueven longitudinalmente a la vez, con lo que todas las cabezas se encuentran en todo momento en la misma vertical. Este hecho nos lleva a definir el concepto de cilindro.

Cilindro: es el conjunto de todas las pistas a las que se puede acceder teniendo el mecanismo de acceso en una posición determinada; o también, conjunto de pistas a las que se puede acceder para un determinado movimiento de brazo. (**figura 2**)

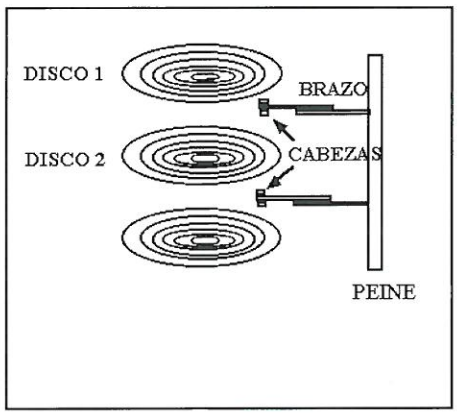


Figura 1. DISPACK.

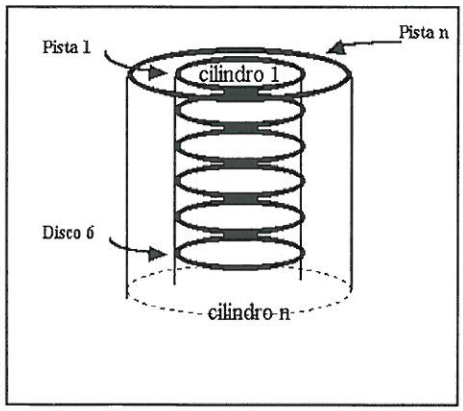


Figura 2. Cilíndro y Pista.

Cada disco tiene un numero determinado de pistas concéntricas, las cuales tienen la misma capacidad, por lo que varía la densidad en cada una de ellas.

La información se graba por cilindro, no por superficie, ganando de esta forma *tiempo de posicionamiento* de las cabezas. Y, dentro de cada pista, la información se graba en serie, bit a bit.

Existen, como el lector puede suponer, diferentes modelos de discos entre los que se pueden citar a modo de ejemplo los siguientes:

UNIDAD IBM 2311

Cada *dispack* tiene 6 discos, de los cuales son útiles solo 10 superficies de las 12 de que dispone. No se graba información ni en la cara superior del conjunto ni en la inferior, porque es de suponer que en estas caras se concentran impurezas del ambiente, tales como polvo, rayas, etc.
Tiene 203 pistas, y en consecuencia 203 cilindros.

La capacidad de una pista es de 3625 octetos y por tanto la de un cilindro será 3625 * 10.

UNIDAD IBM 3330

Cada dispack o pila de discos contiene 11 discos, de los cuales sólo se utilizan 19 superficies. Cada superficie tiene 411 pistas siendo la capacidad de una pista de 13030 octetos.

UNIDAD IBM 3340

En esta unidad el dispack, que se compone de 6 discos, es hermético por lo que se utilizan las 12 superficies de los discos.

El modelo M35 de esta Unidad dispones de 349 pistas por superficie, siendo la capacidad de una pista de 8368 octetos.

El modelo M70 contiene 701 pistas por superficie, y la capacidad por pista es de 8368 octetos.

ORGANIZACIÓN DE FICHEROS

Método de organización : Es el conjunto de técnicas establecidas para almacenar la información soportada en un volumen o soporte de información. Existen 5 métodos, tal y como se muestra en la figura 3, utilizándose uno u otro, por lo general, en función de cómo las aplicaciones van a tratar la información contenida en los distintos ficheros.

Método de acceso: Es la técnica utilizada para almacenar, localizar y recuperar de un soporte la información auto-

Metodo de organizacion	Metodo de acceso
Secuencial	SAM (Sequential Access Method)
Secuencial indexado	ISAM (Indexed Sequential Access Method)
Directa	DAM (Direct Access Method)
Particionada	PART (Particioned)
VSAM	VSAM (Virtual Storage Access Method)

Figura 3.

máticamente. También hay 5 tipos, dependiendo de la organización del soporte.

El cuadro de la **figura 3** muestra la relación entre métodos de organización y métodos de acceso.

FORMATO DE LOS REGISTROS

Los archivos en MVS pueden estar formados por registros de longitud FIJA (LF) o VARIABLE (LV), estando en este caso precedido por 2 bytes, en binario puro, indicando la longitud del registro.

Así mismo, tanto unos como otros pueden agruparse o no en bloques, para grabarse en el volumen como BLOQUEADO (B) o NO BLOQUEADO (NB).

Por ultimo, existe otro formato, INDEFINIDO, que es usado normalmente para el almacenamiento de los módulos ejecutables.

Existen por tanto las 5 modalidades siguientes de formatos de registro: RLFNB, RLFB, RLVNB, RLVB y RLU.

VTOC: (Volume Table Of Contain)

El VTOC es un archivo especial que obligatoriamente se encuentra en todos los volúmenes de la instalación, y que contiene los nombres de todos los ficheros que se encuentran en ese volumen. Este fichero recibe también la denominación de archivo de etiquetas y se encuentra en un área especifica del volumen. Se podría comparar con la FAT (File Allocation Table) de los disquetes del MS-DOS.

El propio VTOC, por ser un fichero, también deberá estar referenciado dentro de sí mismo y tener por tanto su propia etiqueta, que precisamente se llama VTOC.

Esta etiqueta constituye el primer registro del fichero VTOC, el cual tiene un registro con formato especial, pues contiene información propia del VTOC. Este fichero puede estar en cualquier parte del volumen, y consta de un numero entero de pistas (dentro de un cilindro) llegando como máximo a un cilindro. Las pistas no son compartibles.

Este archivo se crea al inicializar cada volumen, pues el programa que realiza esta función, entre otras cosas se encarga de:

- Grabar la dirección del VTOC en la etiqueta de volumen
- Grabar la etiqueta de VTOC en la primera posición del VTOC
- Generar la segunda etiqueta del VTOC como etiqueta de espacio disponible.
- Reservar el espacio o área restante del VTOC.

Los registros del VTOC son registros con clave formados por los dos campos o áreas siguientes:

- área de clave de 44 octetos que contendrá el nombre del fichero.
- área de datos de 96 octetos, que contendrá características, estadísticas y el puntero a comienzo del fichero.

Existen distintos formatos de registros de VTOC, identificados por los valores claves F1 a F6, pero que si bien contienen información importante del VTOC, no es relevante para el objetivo de este articulo.

El VTOC es un archivo especial que obligatoriamente se encuentra en todos los volúmenes de la instalación

Organización SECUENCIAL

En esta organización, también llamada CONSECUTIVA, la información se almacena colocando cada registro uno a continuación de otro, como su propio nombre indica, y tiene la particularidad de que para recuperar una información se debe leer necesariamente toda la anterior.

Este tipo de organizacion es una organizacion totalmente compacta, es decir, sin huecos entre sus registros.

Los formatos admitidos son de longitud fija y variable, bloqueados y no bloqueados.

Procesos que se pueden realizar con estos ficheros:

Carga: La carga se debe realizar mediante procesos secuenciales.

Recuperación: La recuperación de un registro supone la lectura consecutiva de todos los registros que preceden al registro a recuperar, debiendo realizarse mediante procesos secuenciales. Lo normal es que los registros lleven secuencia de clave.

Existe un índice que sirve para medir el tiempo medio de acceso a un registro. Es el tiempo de registro, y se define como t=T/n

donde t representa el tiempo medio de proceso de registro

T representa el tiempo total de proceso de fichero

n representa el numero total de registros.

Actualización: Los procesos de Alta - Baja y Modificación requieren el uso de un fichero auxiliar sobre el que se irán reflejando las actualizaciones que se requieran, y que deberá ser consultado en cada uno de los accesos que se realicen al fichero maestro a fin de determinar si los datos del registro principal han sido modificados (FIGURA 4-A). Esta consulta será necesario realizarla hasta que se realice el proceso de actualización, mediante el cual se fusionara el fichero maestro y el fichero auxiliar, generándose un fichero nuevo actualizado, para posteriormente borrar o pasar a histórico el maestro anterior (FIGURA 4-B).

Cuando se da secuencialidad a un fichero mediante el uso de una clave, se pasa de una organización consecutiva a una organización secuencial.

Organización SECUENCIAL-INDEXADA

Este tipo de organización tiene las siguientes particularidades:

- Todos los registros se encuentran clasificados en orden ascendente de clave.
- Para poder recuperar un registro determinado mediante un proceso al azar, es preciso realizar un estudio jerárquico de los índices que se definan. Estos índices son unos registros especiales que contienen la información necesaria para permitir el acce-

so directo al registro sin necesidad de recuperar todos los registros previos.

Antes de tratar esta organización, y con objeto de facilitar la explicación correspondiente, el autor propone el siguiente ejemplo:

Imaginemos un diccionario que por ser demasiado voluminoso, se encuentra dividido en tres tomos, figurando en el lomo de cada tomo la primera y última palabra descrita en el mismo (**figura 5**)

Por su parte, cada página de ese diccionario dispone en su parte superior de la ultima palabra que se encuentran en esa página.

Por último, también existe un cuarto volumen, el APÉNDICE, en el cual debemos buscar aquella palabras que se han definido con posterioridad a la edición del diccionario.

Pues bien, si el lector quisiera conocer el significado de la palabra *FICHERO*, debería buscar en el lomo del Tomo-I y al comprobar que CALA, alfabéticamente, es anterior a FICHERO, desecharía dicho volumen y pasaría a examinar el Tomo-II, que como contiene las palabras desde CALAR a PISTO debe incluir la que se busca.

A continuación esa misma selección se haría con las páginas, no fijándose en el contenido de las mismas sino en su índice superior, hasta que llegue a la pagina 257, en la cual resulta que FIEL es la primera palabra del índice que es posterior a FICHERO, y como a su vez FICHA es anterior, resulta que la definición buscada está en esa pagina 257.

Una vez seleccionada la página, volvemos a hacer la misma operación, pero esta vez con las palabras de la pagina, hasta encontrar FICHERO.

Pues bien, con la organización secuencial indexada, el propio sistema en tiempo de generación del fichero, cuando va colocando la información en el área de datos reservada al efecto, va generando en el área de índices unos índices jerarquizados de acuerdo con el campo clave de esa información y que, al igual que en el ejemplo visto, consta de los siguientes índices:

- *Índice MAESTRO :* representaría el índice de la biblioteca.
- *Índice de CILINDRO:* representado por la información contenida en el lomo de cada tomo
- *Índice de PISTAS:* Estaría representado por las palabras de cabecera de cada página.

Y por último, el dato buscado se encontraría examinando secuencialmente la clave de todos los registros de información contenidos en ese bloque o pagina seleccionada.

Si al tiempo que la Real Academia va creando una palabra, pudiera poner una marca en el hueco donde debiera ir para remitirnos al apéndice, nos habría resuelto el tener que explicar el hecho de que el índice de pistas tenga dos entradas, :

- la entrada normal; y
- la entrada de excedentes que se vera más adelante.
-

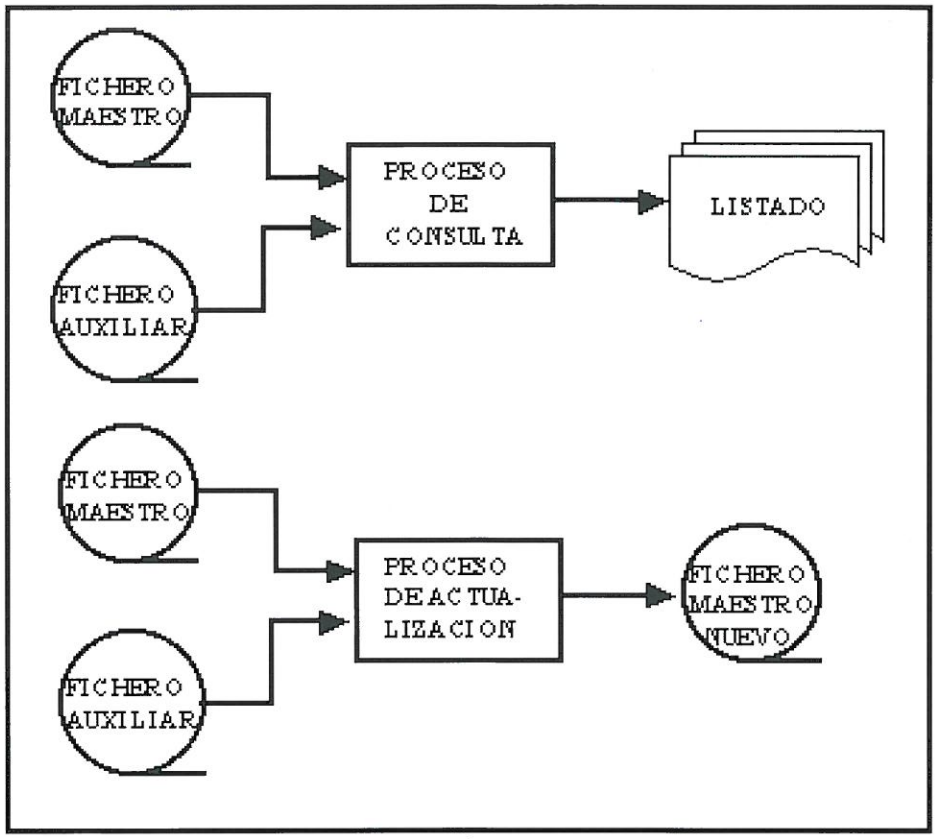


Figura 4.

Índices

Los índices son ficheros especiales, compuestos por unos registros llamados entradas y que siempre estarán en formato desbloqueado aunque el área de datos este bloqueado.

Estas *entradas* tienen dos campos o áreas: Área de clave y área de datos.

En el área de índices, la clave tendrá la misma longitud que en el área de datos; y en el área de datos del registro de índices figurará un puntero con una longitud de 10 octetos.

Los índices son generados por el método de gestión de los datos de forma automática, es decir, el método ISAM va generando los índices al tiempo que se crea el fichero.

Como ya se ha dicho anteriormente, este método cuenta con los TRES ÍNDICES siguientes:

El índice de pistas es obligatorio e ira siempre en la pista 1 de cada

Los índices son generados por el método de gestión de los datos de forma automática

cilindro asignado al fichero, reservándose el resto del cilindro para datos del fichero, y dejando normalmente al ultima pista como área de overflow.

La información que lleva este índice de pistas hace referencia a cada una de las pistas de ese cilindro, tal y como se puede apreciar en la **figura 6**.

El índice de pistas es el de mas baja jerarquía, pues remite directamente a la pista donde se encuentra la información. Un fichero puede tener asignados una o varias pistas para ser utilizadas como índices de pistas de cada cilindro, aunque lo normal es que haya sólo una.

Cada pista del área primaria genera dos entradas en el índice de pistas:

- una entrada normal que contendrá la dirección de la pista y la clave mas alta de esa pista; y
- una entrada de excedentes, que contendrá la entrada de excedentes, es decir en el área de clave figurará la clave mas alta de la cadena asociada, y en el área de datos, un puntero con la dirección de la pista(no del registro) donde se encuentra el registro de clave mas baja de la cadena de excedentes asociada a la pista de entrada normal.

Índice de cilindros: Este índice también es obligatorio. Contiene una entrada por cilindro, y puede grabarse en cualquier pista del dispack, o incluso en otro dispack.

La información que lleva es:

- en el área de clave la clave más alta del cilindro; y
- en el área de datos, la dirección del cilindro.

Índice maestro: Este índice es optativo. Es un índice de pistas de un índice de cilindros. Se utiliza cuando el índice de cilindros ocupa más de 4 pistas.

Las entradas de este índice contienen la siguiente información:

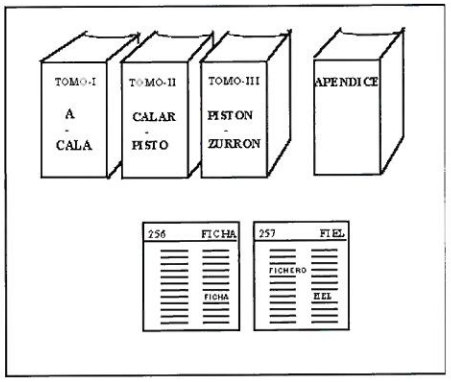


Figura 5.

En el área de clave: la clave más alta de la pista del índice maestro

En el área de datos: la dirección de esa pista.

Cadena de excedentes

Los registros en el área de overflow se encuentran formando cadenas de excedentes.

Una cadena de excedentes es un conjunto de registros que deberían pertenecer a una pista determinada de datos, pero que por haberse dado de alta con posterioridad a la carga inicial del fichero se han encontrado sin espacio físico donde ubicar esa información. Esto quiere decir que una cadena de excedentes esta asociada a una pista de datos.

Los registros en el área de overflow se encuentran en formato desbloqueado, y el propio sistema operativo añade a cada registro un campo de enlace o puntero de 5 octetos, en el cual coloca la dirección del siguiente registro lógico, o 'FF' hexadecimales si es el último registro de la cadena.

En tiempo de creación, las áreas de overflow están vacías. Con el uso, al irse llenando, la flexibilidad del fichero disminuye. En el caso de que se omita en tiempo de creación la definición de este área, el sistema reserva un determinado numero de pistas del cilindro.

En el área de overflow se colocan, no los registros nuevos, sino los que sobran de las pistas del área primaria al colocar en su sitio al registro nuevo.

Si por casualidad el área de overflow de un cilindro quedara llena, y hubiera que colocar un registro excedentario, este se colocaría en el área de overflow independiente.

Hay que tener en cuenta que en el caso de que una pista de datos cuente por ejemplo con tres registros: de clave 19, 45 y 87 en el índice de pistas viene la clave del registro mas alto y la dirección del registro mas bajo, es decir: 87-d(19)

También hay que tener en cuenta que en el área de overflow, los registros cuentan con un campo llamado campo de enlace, en el cual figura la dirección del siguiente registro lógico.

La información contenida en el campo de enlace la genera y la trata el propio sistema.

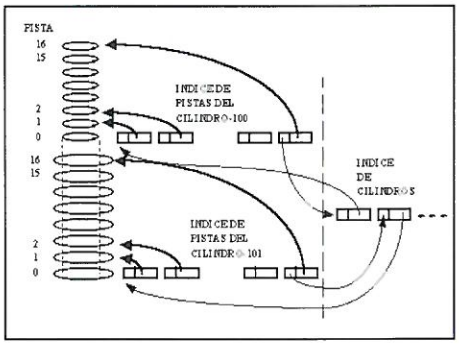


Figura 6.

La información contenida en el campo de enlace la genera y la trata el propio sistema

PRÓXIMA ENTREGA

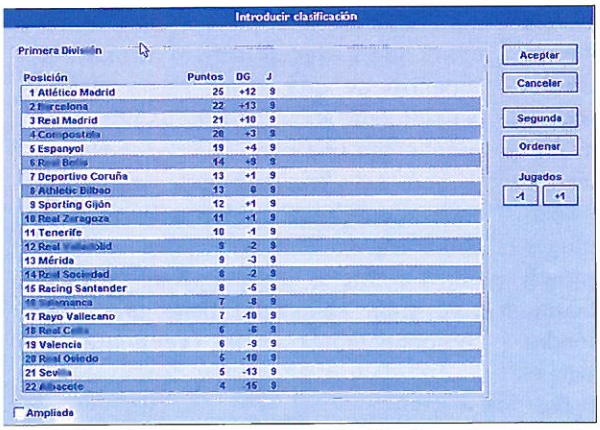
Evidentemente el tema es largo por lo que debe quedarse para la próxima entrega las Organizaciones Directa y VSAM, así como unos ficheros especiales que juegan un papel muy importante en el acceso a la información de los ficheros: son los Catálogos. Por ultimo, también se comentarán los ficheros GDG y si es posible, ejemplos de definición de cada uno de los tipos descritos.

SUPER QUINIELAS 3.0

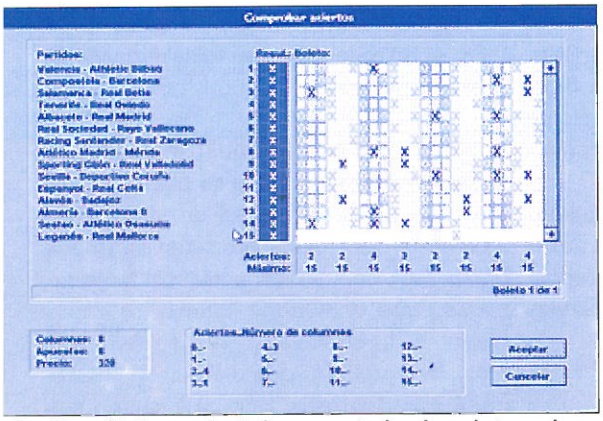
TRES TEMPORADAS FABRICANDO MILLONARIOS

1993 el comienzo de Super Quinielas.

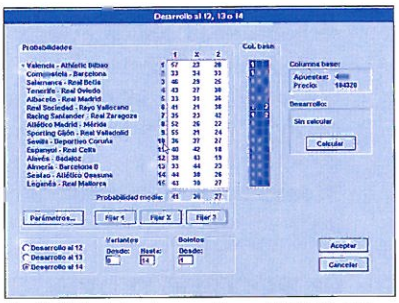
1994 segunda versión del programa.



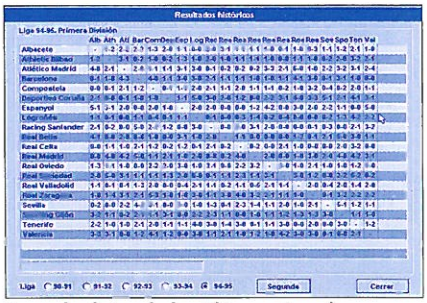
Siga la liga de primera y segunda división jornada a jornada, con las opciones de Super Quinielas.



Identifique fácilmente los boletos premiados de cada jornada.



Posibilidad de desarrollos al 12,13 ó14.



Base de datos de las últimas cinco ligas.

- *Tres sistemas de cálculo diferentes: histórico, últimos resultados, y por clasificación actual.*
- *Reducción al 12, 13 y 14.*
- *Base de datos de consulta de los últimos 5 años: primera y segunda división.*
- *Compruebe sus boletos premiados de la manera más rápida.*
- *Ideal para las peñas quinielísticas.*
- *Permite apostar mediante disquete.*
- *Válido para los próximos años / opción de nueva liga.*
- *Dos versiones por el precio de una: DOS y WINDOWS*

CON LA GARANTÍA DE DDM DIGIT AL DREAMS MULTIMEDIA

Solicita SUPER QUINIELAS 3.0 enviando este cupón o llamando al teléfono (91) 661.42.11* de 9 a 14 y de 15:00 a 18:00 o por Fax: (91) 661.43.86

Deseo que me envíen: ☐ **SUPER QUINIELAS 3.0 por 2.495 ptas. + 250 ptas. de gastos de envío.**

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....

Teléfono .....F. de nacimiento.....Profesión.....

FORMA DE PAGO:

☐ Talón a ABETO EDITORIAL ☐ Contra-reembolso

☐ Giro Postal nº.....de fecha.....

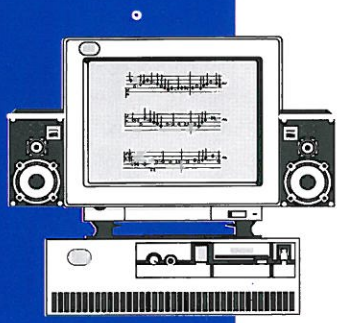
☐ Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ CADUCA ☐☐ ☐☐

Firma ,

ABETO

Rellena este cupón y envíalo a:
ABETO EDITORIAL
C/ Aragoneses, 7.
28100 Pol. Ind. ALCOBENDAS (Madrid)

FILTROS DIGITALES EN TIEMPO REAL



Daniel Pérez

En pasados artículos de esta revista se han tratado temas referentes a la reproducción del sonido (proveniente de un fichero .VOC, .WAV ó .MOD) y, también, a la digitalización del mismo (consiguiendo así grabar sonidos en ficheros). Todo ello programando adecuadamente la Sound Blaster (también se explicó cómo programar la Gravis UltraSound).

En el presente artículo se supondrá que sabemos digitalizar y reproducir el sonido, para centrarnos, así, en el problema que nos concierne: el apasionante mundo del procesado del sonido (o tratamiento de la señal).

Para entender adecuadamente el programa, sería recomendable disponer del artículo nº 7 de programación de la Sound Blaster (revista nº 14), en el que se habla de la digitalización y reproducción del sonido).

Quien no esté interesado en la justificación teórica del proceso puede pasar directamente a los apartados: "Convolución" y "Cálculo de coeficientes".

Todos hemos visto representado el espectro de una señal en las cadenas de música, es lo que se llama transformada de Fourier y es otra manera de representar la misma señal. También estamos acostumbrados a ecualizar el sonido con unas barritas que subimos y bajamos, no son más que filtros digitales en tiempo real.

UN POCO DE TEORÍA

En primer lugar se describirán conceptos básicos sobre señales analógicas. Y, en segundo lugar, las señales digitales o discretas, que son las que se pueden tratar digitalmente con un PC o con un DSP (Digital Signal Processor). Un DSP no es más que un microprocesador especializado en el tratamiento de señales, incluido en cualquier cadena de música.

Uno de los conceptos más usados en el campo del tratamiento de la señal es la famosa **transformada de Fourier** o, para entendernos mejor (aunque en teoría no es exactamente lo mismo), el **espectro** de una señal. Hasta ahora, simplemente nos imaginábamos una señal como una onda que dependía del tiempo, pero con este nuevo concepto podemos representar las señales fijándonos en su distribución frecuencial. Como ejemplo, el espectro de un tono puro (una sinusoide) tiene una sola componente frecuencial. (Ver figura-1).

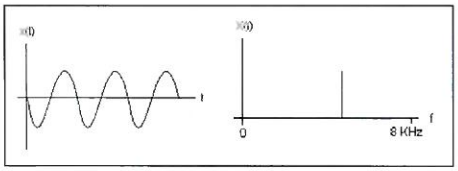


Fig. 1. Un tono puro y su tranasformada de Fourier (espectro).

Y, como se sabe que, matemáticamente, cualquier señal se puede expresar como suma infinita de sinusoides, tenemos que cualquier señal se puede expresar con su espectro en frecuencias, que contendría infinitas componentes frecuenciales (evidentemente, existen múltiples demostraciones matemáticas rigurosas, pero nos saldríamos del tema de esta revista). (Ver figura-2).

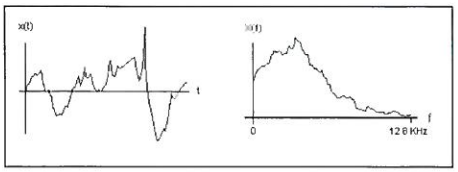


Fig. 2. Una señal cualquiera con su correspondiente representación espectral.

Existe una fórmula analítica, con la cual podemos hallar el valor (amplitud,

potencia) de cada componente frecuencial:

$$X(f) = TF\{x(t)\} = \int_{-\infty}^{+\infty} h(i)\Diamond\exp(-j2pfi)di$$

Nota: La transformada de Fourier de una señal x(t) se suele representar como: TF{x(t)} o X(f).

Para salir de dudas, el espectro de una señal es algo a lo que todos nosotros estamos muy habituados a observar: en las cadenas de música hay unos dibujos en forma de barritas que no paran de subir y bajar, si nos fijamos, podremos ver que cada barrita representa un margen de frecuencias (si quisiéramos poner una barrita para cada frecuencia, necesitaríamos una cadena de música infinitamente grande, ya que, la frecuencia es un parámetro real y, aunque sólo representáramos un margen finito de frecuencias, en él habría contenidas infinitas frecuencias). Como mínimo suele haber 5 barritas, teniendo una frecuencia central cada una de: 100Hz, 330Hz, 1khz, 3.3khz, 10khz aproximadamente.

En concreto, la voz humana tiene su mayor contenido espectral en un margen de frecuencias que va de los 300Hz a los 3400Hz (margen que es el que, precisamente, se transmite a través de la red telefónica, ya que, es una red pensada para la voz). Aunque cabe notar que fuera de este rango de frecuencias existen componentes frecuenciales en la voz humana que, aunque conscientemente no oímos, sí juegan un papel importante a nivel inconsciente en cuanto al reconocimiento de la persona que oímos hablar, además, según qué personas no se ajustan del todo a este rango (por eso, a veces nos cuesta reconocer a algunas personas por teléfono).

Recordemos que ya sabemos expresar una señal en dos dominios: en el tiempo y en frecuencia.

Algo que también es importante saber, aunque no lo demostraré ni siquiera intuitivamente es el teorema de la convolución: la convolución (cuyo símbolo es *) en el tiempo es equivalente al producto en frecuencia (que representaremos como ◊):

si x(t)=a(t)*b(t) entonces tenemos que TF{x(t)}=TF{a(t)}◊TF{b(t)}

A continuación vamos a dejar el mundo analógico (que no nos interesa) y pasaremos al mundo digital, discreto, que es el que tenemos una vez hemos digitalizado la señal con la Sound Blaster o similares. En estos momentos, ya no tenemos una onda continua (x(t)), sino que tenemos una ristra de números (x{n}), estos valores son muestras de la señal original que se toman a una determinada frecuencia: la llamada frecuencia de muestreo (Fm). Evidentemente, cuanto mayor sea la frecuencia de muestreo mayor será la calidad, por tanto, podríamos pensar que lo mejor es digitalizar a la mayor frecuencia posible, pero entonces, la cantidad de datos que obtendríamos sería muy elevada y nos ocuparía mucho espacio. Tenemos, pues, que tomar una decisión adecuada entre un claro compromiso entre calidad y espacio de almacenamiento (o consumo de CPU si hablamos de tiempo real, tema en el que entraremos en detalle más adelante). La solución a este dilema nos la da el **teorema de Nyquist**, el cual nos dice que para poder reconstruir una señal digitalizada a su señal original, nos basta con que la frecuencia de muestreo sea el doble del el ancho de banda de la señal a digitalizar. En nuestro caso (voz), el doble de la frecuencia máxima sería en teoría de 6800Hz, pero se suele tomar una Fm de 8Khz, así cogemos un ancho de banda de 0Hz a 4Khz. Para la música se suele tomar un ancho de banda de 22 KHz aproximadamente, lo que implica una Fm=44KHz.

Una vez tenemos la señal digitalizada es cuando, desde el ordenador, podemos tratar esta señal discreta, filtrarla e, incluso, hallar su espectro. Para hallar el espectro de una señal discreta se usa la Transformada de Fourier Discreta (Discrete Fourier Transform), DFT:

$$X[k] = \sum_{n=0}^{n=N-1} x[n]\Diamond\exp(-j2pkn/N)$$

donde: N es el nº de muestras que tomamos (cuanto mayor sea más exactos serán los cálculos, pero más lento será el proceso.
x[n] son las muestras de la señal temporal.
X[k] son las muestras del espectro de la señal.

La fórmula anterior es la dual de la transformada de Fourier analógica.

Aunque en la práctica no se usa la DFT por ser muy lenta (y no permitir programas en tiempo real) y, en su lugar, se usa una fórmula equivalente pero muy optimizada: la Transformada Rápida de Fourier (Fast Fourier Transform) FFT (fórmula que revolucionó el análisis de señales digitales enormemente, ya que, permitía su ejecución en tiempo real). En el disquete se incluye un programa que calcula el espectro de una señal. Con este programa podemos hacer pruebas y observar que cuando emitimos un tono más o menos puro, su espectro es un pico en una frecuencia y cero el resto.

Filtrar un sonido no es más que sumar ponderadamente a cada muestra N muestras

CONVOLUCIÓN

Una última pincelada de teoría sobre el tema en cuestión implica explicar la operación de convolución (en el ámbito analógico la expresión de la fórmula sería equivalente a la que, a continuación tenemos, pero el sumatorio se transformaría en una integral):

$$x[n]*y[n] = \sum_{k=-\infty}^{k=+\infty} x[k]\Diamond y[n-k]$$

donde: x[n] son las muestras temporales de una señal.
y[n] ídem de la otra señal.

Nota: En la fórmula anterior observamos que hay un sumatorio infinito,

```
entradas: x[n], y[n] de longitudes L1 y L2
salida: s[n] de longitud L1+L2-1
bucle externo: n=0 ~ L1-1
{
    suma=0;
    lim_inf = Máximo(0, n-(L2-1));
    lim_sup = Mínimo(L1-1, n);
bucle interno: j=lim_inferior ~ lim_superior
    {
        suma = suma + x[j] ◊y[n-j]
    }
    s[n]=suma;
}
```

Ejemplo: si x[n]= {0,1,2,3,4} e y[n]= {5,6,7,8,9} entonces su convolución valdría:

{0,1,2,3,4} fi s[0] = 0◊5 = 0
{9.8.7.6.5}

{0,1,2,3,4} fi s[1] = 0◊6 + 1◊5
{9.8.7.6.5}

{0,1,2,3,4} fi s[2] = 0◊7 + 1◊6 + 2◊5
{9.8.7.6.5}

{0,1,2,3,4} fi s[3] = 0◊8 + 1◊7 + 2◊6 + 3◊5
{9.8.7.6.5}

{0,1,2,3,4} fi s[4] = 0◊9 + 1◊8 + 2◊7 + 3◊6 + 4◊5
{9.8.7.6.5}

º
º

{0,1,2,3,4} fi s[7] = 3◊9 + 4◊8
{9.8.7.6.5}

{0,1,2,3,4} fi s[8] = 4◊9
{9.8.7.6.5}

operación imposible de realizar, ya que, requeriría un tiempo también infinito. Por eso, en la práctica, se usan aproximaciones para solucionar el problema que más adelante trataremos.

Algoritmo: Si consideramos que las dos señales son nulas para n<0 y para n≥L, entonces la convolución de ambas señales vendría dada por un proceso semejante al siguiente(donde la longitud de la señal resultante es la suma de longitudes de las entradas menos 1):

En efecto vemos que la longitud de la salida es la suma de las longitudes de las entradas menos 1.

DISEÑO DE FILTROS

En primer lugar se debe conocer qué es un filtro. Un filtro no es más que "algo" (más técnicamente una caja negra) que sólo permite pasar a través de él determinadas componentes frecuenciales de una señal.

Así, un filtro paso-bajo será aquel que deje pasar las frecuencias bajas, un filtro paso-alto lo contrario y un filtro paso-banda aquél que deja pasar un margen de frecuencias arbitrarias. Las frecuencias umbrales de estos filtros (frecuencias a partir de la cual se deja pasar la señal) son totalmente arbitrarias. Estos filtros que hemos comentados son los más sencillos, no obstante, existen infinitos tipos de filtros (teniendo en cuenta que no tenemos porqué limitarnos a dejar pasar o no determinadas componentes frecuenciales, sino que podemos colorearlas, es decir, multiplicarlas por diversos valores).

Existen múltiples métodos de diseño de filtros, algunos especiales únicamente para imágenes, otros para sonido... Pero hay 3 que son los más sencillos de implementar (enventanado de la respuesta impulsional ideal, DFT inversa del muestreo de la respuesta frecuencias ideal y filtro óptimos), de los cuales nosotros usaremos el más cómodo, sencillo y adecuado para nuestros propósitos: la DFT inversa del muestreo de la respuesta frecuencial ideal, que explicaremos seguidamente. La idea es sumamente sencilla.

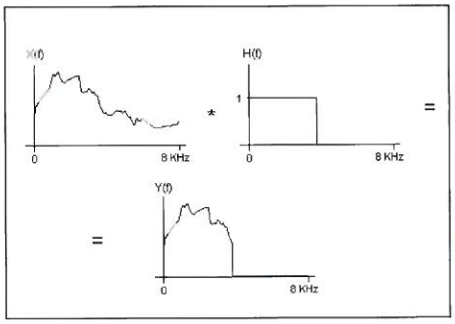


Fig. 3. Proceso de filtrado con espectros de señales (filtro paso-bajo).

Imaginemos que tenemos representado el espectro de una señal y queremos realizar un filtrado paso-bajo (dejar pasar las frecuencias bajas y anular las altas por completo, teniendo en cuenta que el límite de separación lo fijamos nosotros arbitrariamente como se puede observar en el programa de ejemplo). Una posible solución sería multiplicar el espectro original por un pulso de amplitud 1 en el margen de frecuencias que queremos dejar pasar y 0 en el resto (este pulso es, en realidad, el espectro del filtro). (Ver figura-3).

Si recordamos lo explicado en los párrafos anteriores, veremos que un producto en el dominio frecuencial equivale a una convolución en el dominio temporal, por tanto, ya tenemos la fórmula que nos permitirá filtrar:

si Y(f) = X(f) ◊ H(f) entonces tenemos:
y[n] = x[n] * h[n]

Nota: x[n] es la entrada, y[n] es la salida y h[n] representa el filtro.

Es decir, que la salida filtrada no es más que la entrada convolucionada con el filtro. Teniendo en cuenta la fórmula de la convolución (sencilla y fácil de implementar), nos encontramos con tres dificultades: el sumatorio infinito, la anticausalidad del filtro y el cálculo de los coeficientes del filtro.

En cuanto al sumatorio infinito, la solución consiste en aproximar. Si observamos la forma de la transformada inversa de Fourier de un filtro(Ver Figura-4), veremos que, a medida que n aumenta, los coeficientes del filtro (x[n]) disminuyen, por tanto, una buena solución consiste en truncar el filtro, es decir, considerar que a partir de cierta n vale cero, así, el sumatorio infinito se nos ha transformado en un sumatorio finito (en DSP´s, para obtener buena calidad de filtro se usan 32 coeficientes como máximo, pero en general, con 4 u 8 bastará, es cuestión de ir probando con varios números de coeficientes).
Nota: Por el hecho de eliminar ciertos coeficientes, surge un problema adicional, y es que, aparecen componentes frecuenciales que, en realidad, no existen (aunque si se cogen suficientes, este problema es despreciable). Una manera de solucionar esta nueva dificultad es enventanando el filtro en el dominio temporal con alguna ventana tipo Hamming (la que uso en el programa ejemplo) o Blackman, que lo único que hacen es forzar a que los coeficientes del filtro no valgan repentinamente cero en la n de truncamiento, sino que, esto se produzca paulatinamente. (Ver tabla-1).

El problema de la causalidad es muy sencillo de abordar. Si observamos la fórmula de la convolución discreta, veremos que, a cada muestra le tenemos que sumar muestras anteriores y posteriores ponderadas con los coeficientes del filtro, pero si hablamos de

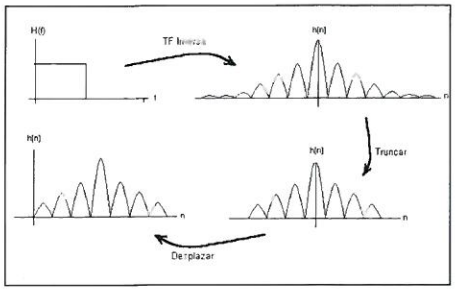


Fig. 4. Representación espectral y temporal de un filtro paso-bajo con las aproxima-

tiempo real, las muestras posteriores no las conocemos porque aún no disponemos de ellas (sería adivinar el futuro).

Para solucionar este problema, basta con forzar a que el filtro tenga coeficientes nulos para n<0, por tanto, podemos desplazar los coeficientes del filtro (con esto estamos introduciendo un retraso entre la entrada y la salida de NumCoeficientes/2 muestras, retraso despreciable dada la frecuencia de muestreo). (Ver Figura-4)

Cabe notar que, en según qué aplicaciones, disponemos de la totalidad de las muestras desde un principio y, por tanto, no hace falta esta aproximación.

El problema del valor de los coeficientes del filtro es también fácil de solucionar. Del filtro conocemos su expresión en el dominio frecuencial (su espectro) y nos gustaría encontrar su expresión en el dominio temporal. Para solucionar este problema usaremos la Transformada Inversa de Fourier Discreta: Inverse DFT:

$$x[n] = \sum_{k=0}^{k=N-1} X[k] \cdot \exp(j2\pi kn/N) \qquad [1]$$

donde: X[k] son las muestras del espectro de la señal
x[n] son las muestras del señal en el tiempo
N es el nº de muestras de la señal (tanto en tiempo como en frecuencia)

CÁLCULO DE COEFICIENTES

En primer lugar veremos la equivalencia entre muestras discretas (k) y frecuencias analógicas (f). N es el nº de muestras en frecuencia (cuanto más exacta queramos la frecuencia umbral del filtro mayor ha de ser). Nosotros

```
int f1,int f2,int Fm;
const int N=256;
 //Leyenda f1 - n1  (min 1)
 //       f2 - n2  (max 128)

 int n,n1,n2,desplaz=(NumCoefs-1)/2;
 float real,num,den,Real,hamming;

 n1=(long)f1*N/Fm;
 n2=(long)f2*N/Fm;

 if(n1==0) n1++;    //El sonido no tiene
componente continua
 if(n1<1 || n2>N/2 || n1>n2) exit(1);
 if(!impar) exit(1); //Esto es por el bucle de abajo (se
puede perfeccionar)

 //Una iteraci¢n del bucle para cada coeficiente
 for(n=0;n<=desplaz;n++)
 {
            real=-2.0*cos(2.0*Pi/N*(n1+n2-1)/2*n);
            num=sin(2.0*Pi/N*(n1-n2-1)/2*n);
            den=sin(2.0*Pi/N*n/2);

            if(num==0 && den!=0) Real=0;
            else if(num==0 && den==0)
            {//Aplico Hoppital para evitar singularidad
            num=cos(2.0*Pi/N*(n1-n2-1)/2*n);
            den=cos(2.0*Pi/N*n/2);
            Real=real*num/den*(n1-n2-1);
                                    }
            else Real=real*num/den;

            hamming=.54
-.46*cos((2*(n+256/2)*Pi)/(N-1));    //Enventanado
Hamming
if(n==0)    plCoefs[desplaz]=Round(Real/N*ham-
ming*ampl_filtro);
else  plCoefs[desplaz+n]=Round(plCoefs[desplaz-
Real/N*hamming*ampl_filtro);
 }
 plCoefs[desplaz]+=Round(ampl_base);
}
```

tomaremos N = 256. Suponiendo que hemos muestreado una señal con una determinada frecuencia de muestreo: Fm, tenemos que la muestra frecuencial k=255 equivale a la frecuencia analógica f=Fm. Por tanto, obtenemos la relación:

$$k = f/Fm \cdot (N-1)$$

Nota: Una propiedad importante que hemos de tener en cuenta es: X[k] = X[N-k] (es por este motivo por el que sólo podremos escoger frecuencias umbral entre f=0 y f=Fm/2, es decir, entre k=0 y k=(N-1)/2).

Los filtros que vamos a diseñar serán: paso-bajo, paso-alto y paso-banda. En estos filtros, dado que simplemente dejamos pasar o no la señal, los coeficientes frecuenciales X[k] de la fórmula anterior sólo podrán tomar los valores 1 y 0.

Imaginemos que queremos un filtro paso-bajo con frecuencia umbral f=2.500Hz (Fm=8KHz), entonces la muestra frecuencial umbral será: k=Round(79.64)=80. Es decir que los coeficientes frecuenciales X[k] valdrán 1 desde k=0 hasta k=80, y cero el resto (teniendo en cuenta la nota previa, también valdrán 1 desde k=256-80=176 hasta k=256-0=256, pero como se sale de los límites tomaremos 255).

Por tanto, el sumatorio de la fórmula [1] (Transformada Inversa Discreta de Fourier) ha quedado reducido a:

$$x[n] = \sum_{k=0}^{k=80} \exp(j2\pi kn/N) + \sum_{k=176}^{k=255} \exp(j2\pi kn/N)$$

Si desarrollamos estos dos sumatorios con coeficientes complejos, veremos que, afortunadamente, las exponenciales complejas se fusionan para dar lugar a un seno real. A continuación podremos observar el algoritmo general resultante:
Notas:

1) k1 valdría 0 en nuestro ejemplo y k2 80.

2) Como sólo queremos un determinado nº de coeficientes, no hace falta que calculemos los 256 y después trunquemos sino que directamente calculamos sólo NumCoefs/2, ya que, después al desplazar se nos multiplicarán por 2.

3) En este algoritmo en seudocódigo no compilable, incluyo la variable ampl_filtro que permite que el filtro no sólo deje pasar ciertas componentes sino que las amplifique.

4) Si nuestra intención fuera aplicar varios filtros en cascada, no haría falta filtrar dos veces, bastaría con sumar/multiplicar a los coeficientes de un filtro los de otro filtro. La función encargada de generar los filtros en el programa del disquete en concreto, recibe como argumentos dos amplitudes, una es la amplitud del filtro paso-banda, y otra es la amplitud por la que

se multiplicarán el resto de componentes frecuenciales (que podría ser 0 y así estaríamos en el caso estudiado hasta el momento). En este caso hay que sumar los coeficientes. De este modo, el programa del disquete nos permite no sólo eliminar ciertas frecuencias sino amplificar algunas y dejar el resto intactas.

CANAL PLUS

Este apartado no tiene relación directa con el tema de filtros, sino que se podría considerar como un anexo o "truco" que, en algunas ocasiones, nos puede resultar de gran utilidad como podremos observar seguidamente.

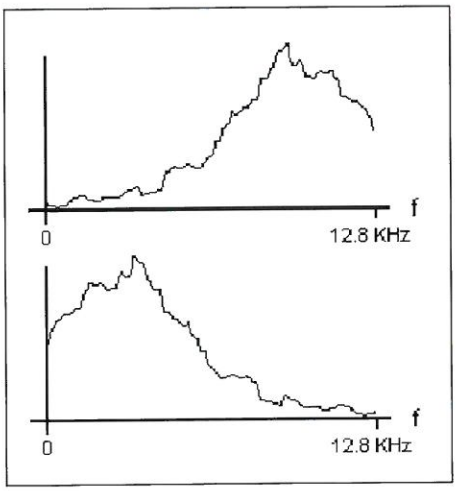


Fig. 5. Espectro del sonido del canal + codificado y decodificado.

Invertir el espectro es un proceso sumamente sencillo, basta con multiplicar cada muestra por cos(Pi.n), que es lo mismo que: +1, -1, +1, -1, +1, -1, +1, -1,...

Por tanto, si cambiamos el signo de una de cada dos muestras, conseguiremos invertir el espectro. Simplemente a modo de curiosidad, es interesante saber que el sonido del canal + está codificado de este modo, con una frecuencia máxima de 12.8KHz, por tanto, si digitalizamos el sonido codificado del canal + con una Fm=25.6KHz (2*12.8khz, por el teorema de Nyquist) o lo más parecido a este valor (a medida que nos alejemos de él la decodificación será peor), y cambiamos de signo una de cada dos muestras, habremos decodificado el sonido del canal +. (Ver figura-5).

EL PROGRAMA

En el disquete se incluyen varios programas, todos ellos en C. RealFFT.exe, RealFilt.exe y CanalP.exe son tres programas para visualizar la FFT, filtrar la señal y decodificar el sonido del canal + respectivamente, todos en tiempo real (sólo funcionarán con versiones de la SB que permitan hacer las conversiones DAC y ADC simultáneamente). FFT.exe, Record.exe, Play.exe, Filter.exe son para visualizar la FFT, grabar (digitalizar), reproducir y filtrar/reproducir las señales, pero no en tiempo real. Al ejecutar cualquier programa sin los argumentos necesarios da un error indicando la manera correcta de ejecutarlo.

También se incluye un fichero de prueba "hola.voc". Es curioso observar la gran diferencia entre ejecutar: "play hola.voc" y "filter hola.voc" con el volumen de la cadena alto. En el primero de los casos se oye la señal original que contiene muchos graves y se hace insoportable. Y en el segundo caso se oye la misma señal pero con un filtro paso-banda entre 450Hz y 4000Hz, con lo cual hemos eliminado los graves tan molestos.

Cabe notar en el programa RealFilt.exe el tamaño del buffer de digitalización y reproducción (constante BufSize), el cual no puede ser muy elevado, porque entonces se produce un notable retardo entre la entrada y la salida (esto significa que nos oímos hablar a nosotros por el micrófono y, al instante, nos volvemos a oír en los altavoces, situación un tanto molesta).

Es importante destacar el hecho de que, únicamente al principio del programa se calculan los coeficientes "mágicos" y, después, simplemente, para cada muestra de la salida, hemos de multiplicar estos coeficientes por N muestras pasadas de la entrada.

Otro aspecto a comentar en el programa es el uso de la técnica del punto fijo para hacer los cálculos. En un principio los coeficientes son números con decimales (floats o doubles), números que, al hacer el filtrado, tendríamos que multiplicar con otros y después sumar. Es ya sabido que, las operaciones con números en notación coma flotante (float, double) son muy lenta en comparación con las mismas operaciones realizadas con enteros (int, long). Por este motivo, lo que se hace es multiplicar estor números con decimales por una constante, por ejemplo 256, redondearlos y, entonces, almacenarlos en forma de enteros. De este modo, al filtrar la señal, simplemente tendremos que multiplicar las muestras por los coeficientes (todas las operaciones son con enteros) y, posteriormente, dividir por la constante inicial para, así, ser coherentes matemáticamente. Todo este proceso se optimiza en velocidad aún más escogiendo como constante una potencia de 2, por ejemplo 256, ya que, así, multiplicar y dividir se transforma en rotar a la izquierda y a la derecha respectivamente.

Invertir el espectro es un proceso sencillo, basta con multiplicar cada muestra por cos(Pi.n)

RESUMEN:

Hemos visto que, a partir de una señal en el tiempo y la aplicación de una fórmula, obtenemos otra representación de la misma señal pero en el dominio frecuencial y no temporal (representación útil para solucionar según qué problemas, como hemos podido comprobar).

También hemos aprendido que, para filtrar, basta con sumar a cada muestra las N muestras anteriores ponderadas con unos coeficientes fijos (sólo se calculan una vez).

IMPLEMENTACIÓN DE UN DEVICE DRIVER (I)

Pere Ridau, Jordi Freixenet, Enrique F. Torres

Los sistemas operativos (S.O.) actuales deben, además de estar diseñados para soportar una gran diversidad de periféricos, prever la incorporación de nuevos dispositivos. Los *device* drivers juegan aquí un papel imprescindible. Podríamos definir un *device* driver como aquella parte del sistema operativo que dialoga directamente con los controladores de dispositivo. El driver actúa de interface software entre el dispositivo y el resto del S.O. Así, siempre que se instale un dispositivo, se deberá añadir el driver correspondiente. En Linux existen dos tipos básicos de drivers:

-drivers de caracteres
-drivers de bloques

En este artículo se explicará qué es un driver *de caracteres* y cómo debe ser programado. El estudio de los drivers de bloques y el proceso de instalación se presentará el próximo mes.

FICHEROS DE DISPOSITIVO

En Linux, al igual que en el resto de sistemas Unix, los dispositivos de entrada/salida están mapeados sobre el sistema de ficheros. Es decir, para cada dispositivo físico conectado al ordenador, existe un fichero asociado, ubicado dentro del sistema de ficheros, normalmente en del directorio /dev. Así por ejemplo el fichero /dev/console identifica a la consola del ordenador, y el fichero /dev/fd0 identifica la primera unidad de disco flexible. Los ficheros de dispositivos pueden ser manipulados de igual manera que los ficheros normales. Una operación de escritura sobre un fichero de dispositivo es traducida por el *kernel* (núcleo del S.O.) en una operación de escritura sobre el dispositivo asociado. De forma paralela, un operación de lectura sobre un fichero de dispositivo se convierte internamente en una operación de lectura desde el dispositivo asociado. A continuación se presentan algunos ejemplos:

$ echo hello world > /dev/lp

Imprime por la impresora el mensaje "hello world".

$ od -c /dev/fd0

Muestra por el terminal el contenido del disco situado en la unidad 0 en formato ascii.

Como ficheros especiales que son, los ficheros de dispositivo no tienen asignados bloques de datos, de manera que únicamente ocupan un *inode* y una entrada de directorio. Podemos diferenciar dos tipos:

-Dispositivos de bloques: se denominan así ya que la transferencia de información entre el dispositivo y el kernel se realiza bloque a bloque, a través de la *cache* de disco. Son dispositivos que pueden contener un sistema de ficheros.

-Dispositivos de caracteres: no pueden contener sistemas de ficheros y la trasferencia de información no pasa por la cache de disco. La principal diferencia con los drivers de bloques, radica en la interface entre el driver y el S.O. (cuándo y cómo el S.O. llama al driver).

CONSTRUCCIÓN DE UN FICHERO DE DISPOSITIVO

Internamente, Linux identifica a un dispositivo a partir del par (*major, minor*). A continuación se presenta el significado de estos valores:

Los *device drivers* son, posiblemente, la parte del sistema operativo más susceptible de ser implementada por un programador. Iniciamos este mes una serie de dos artículos para describir el funcionamiento de los drivers en Linux.

-major: Identifica el tipo de dispositivo. Permite al S.O. localizar internamente las rutinas aplicables a ese dispositivo.
-minor: Permite diferenciar dos dispositivos de un mismo tipo (igual *major*), compartiendo las mismas rutinas del driver, pero operando sobre puertos de entrada/salida diferentes (El *minor*, permitirá, por ejemplo, identificar el puerto a utilizar).

El par (*major,minor*) es único para cada dispositivo del sistema, y está almacenado dentro del inode (ver Fig. 1).

La orden *mknod* permite crear un nuevo fichero de dispositivo con un determinado valor de *major* y *minor*:
$ mknod /dev/mbx0 c 30 1

Crea un fichero de dispositivo denominado *mbx0*, de caracteres, *con* major=30 y minor=1.
$ mknod /dev/rd b 5 1

Crea un fichero de dispositivo llamado rd, de bloques, con major=5 y minor=1.

OPERACIONES UNIFORMES

Linux ofrece un conjunto de llamadas al sistema para operar con ficheros y dispositivos. La Fig. 2 presenta una breve descripción de estas operaciones (para mayor información sobre el tema, consultar la sección 2 del manual de Linux). Dado que las operaciones especificadas anteriormente pueden operar de forma indistinta con ficheros ordinarios o con ficheros de dispositivos, en lo sucesivo se denominaran conjunto de operaciones uniformes.

La integración de los dispositivos dentro del sistema de ficheros, y la existencia de un conjunto de operaciones uniformes para operar de forma indistinta con ficheros y/o dispositivos, es el pilar sobre el cual Linux puede implementar la independencia de dispositivos; esta característica permite a un mismo programa, en diferentes ejecuciones, operar de igual manera con diferentes dispositivos. La Fig. 3 muestra el código de un programa que, utilizando las operaciones uniformes, puede operar de forma indistinta con ficheros y/o dispositivos.

Una operación uniforme, por ejemplo *write*, debe ser aplicable tanto a un fichero ordinario, como a un dispositivo de terminal, o a una impresora. No obstante, es obvio que en última instancia una operación de escritura al terminal, tiene poco que ver con una a la impresora. Seguramente, la primera provoque la salida de información por un puerto serie, mientras que la segunda realice una comunicación de información por el puerto paralelo. ¿Cómo consigue el S.O. implementar las operaciones uniformes?

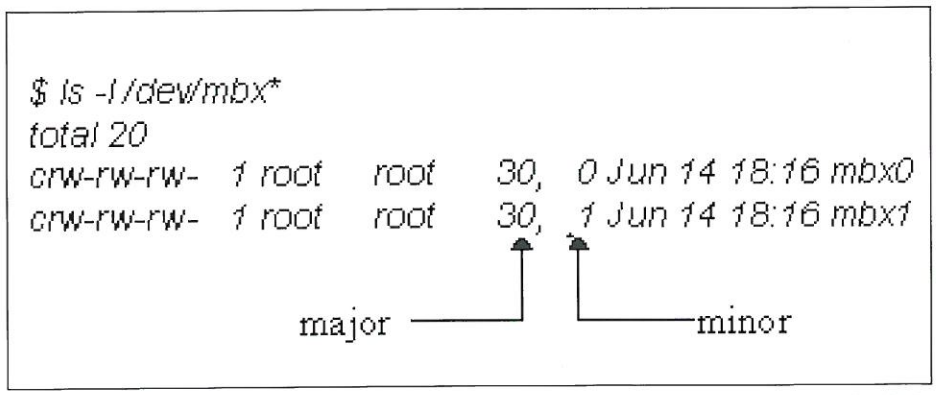


Fig. 1 Se puede examinar el major y el minor de un fichero de dispositivo con la orden ls-l.

OPERACIONES UNIFORMES CON DISPOSITIVOS DE CARACTERES.

Cada operación uniforme (*read* por ejemplo) recibe como parámetro un *descriptor de fichero* (fd, resultado de la operación *open*). Esta operación, provoca en última instancia, una lectura del dispositivo físico asociado a fd. Dado que la operación de lectura física, depende del controlador de dispositivo afectado, el kernel deberá disponer de una rutina específica de lectura, que denominaremos *read_dev*, para cada dispositivo. Cuando un proceso de usuario ejecuta la llamada al sistema *read(fd,buff,n)*, el kernel transferirá el control a la rutina *read_dev* vinculada al driver al cual hace referencia el fd. La rutina *read_dev*, es la encargada de implementar la lectura sobre el dispositivo.

De esta manera, para cada operación uniforme, se dispondrá dentro del kernel de:

- Una rutina específica (ej: *read_dev*) para cada tipo de dispositivo, que implementa la funcionalidad de la operación uniforme asociada dentro del kernel.
- Una rutina genérica, independiente de los dispositivos, perteneciente al kernel, que opera sobre descriptores

· int open(char * pathname, int flags, int prot)
Abre el fichero/dispositivo identificado por pathname, en modo lectura y/o escritura (según el valor de flags), y opcionalmente lo crea (en el caso de que no existiese previamente), con las protecciones especificadas en prot. Retorna un descriptor de fichero que permitirá referenciar el fichero en próximas operaciones o -1 en caso de error.

· int read(int fd, char * buff, int n)
Lee n bytes de información del fichero/dispositivo asociado al descriptor fd, a partir de la posición actual del puntero de lectura/escritura y dejando la información en memoria a partir de la dirección de memòria buff. Esta operación retorna el número de bytes realmente leidos o -1 en caso de error.

· int write(int fd, char * buff, int n)
Escribe los n bytes situados en memoria a partir de la dirección buff, en el fichero/dispositivo identificado por fd, a partir de la posición actual del puntero de lectura escritura. Retorna el número de bytes realmente escritos o -1 en caso de error.

· int lseek(int fd, int offset, int whence)
Sitúa el puntero de lectura/escritura asociado, en la posición especificada por la variable offset. Whence indica como debe interpretarse el nuevo valor del puntero: relativo a la posición actual, relativo al final del fichero, o relativo a la posición actual.

· int ioctl(int fd, int request, char *argp)
Ejecución de una operación atípica sobre el dispositivo asociado a fd. El parámetro request especifica la operación a realizar, y su interpretación depende del dispositivo. El parámetro argp es un puntero a una estructura que contiene los parámetros asociados a la operación request.

· select y mmap
Dado el carácter divulgativo del artículo, obviaremos estas dos operaciones. El lector interesados puede consultar la sección 2 del manual de Linux.

Fig. 2 Llamadas al sistema para operar sobre ficheros/dispositivos.

de fichero, y que implementa la respectiva llamada al sistema (ej: *sys_read*). Ésta, utilizará las estructuras de datos internas del kernel, para acabar llamando a la rutina específica del dispositivo implicado (read_dev).

ESTRUCTURAS DE DATOS INTERNAS PARA LA GESTIÓN DE LA E/S.

Para poder entender cómo el kernel puede enlazar una operación uniforme con la operación específica del dispositivo, es conveniente analizar previamente las estructuras de datos internas de Linux, en las cuales se apoya la gestión de las operaciones de E/S.

Para cada proceso del sistema, existe dentro del kernel una estructura *task_struct* que contiene la información relevante de dicho proceso. Esta estructura contiene un array de descriptores de fichero (FDT), el tamaño del cual coincide con el número máximo de ficheros abiertos que puede gestionar dicho proceso. Cuando un proceso abre un fichero (open), el S.O. realiza las siguientes operaciones:

-Crea e inicializa una nueva estructura file que contiene información referente a una sesión de trabajo con un fichero/dispositivo. Dentro de esta información cabe destacar el modo de acceso, el puntero de lectura/escritura, un puntero al inode del fichero/dispositivo abierto, y un puntero a la estructura *file_operations* que contiene, para cada operación uniforme, la dirección de la rutina específica asociada al dispositivo abierto.

- Busca un elemento libre dentro de la FDT, comenzando desde el elemento 0. Tras encontrarlo, hace que éste apunte a la estructura file creada anteriormente, y retorna como fd el índice a la FDT.

La Fig. 4 muestra las estructuras de datos comentadas anteriormente así como las relaciones existentes entre ellas.

Para cada operación *open* ejecutada existe una estructura file que define la sesión de trabajo de este fichero/dispositivo. Las estructuras file se hallan encadenadas entre ellas formando una lista única dentro del S.O. que denominaremos LFA (Lista de Ficheros Abiertos). Es posible que dos procesos diferentes abran un mismo fichero, e incluso que un mismo proceso abra un fichero dos o más veces. Todas las estructuras file que hacen referencia a un fichero concreto apuntan al inode de éste, que se halla de forma única en memoria. Las inodes de los ficheros que se hallan abiertos en un momento determinado, se encuentran cargados en memoria y encadenados entre ellos formado lo que denominaremos LIA (Lista de Inodes Activos).

El lector interesado en profundizar en el conocimiento de las estructuras de datos internas, puede referirse a [bach86].

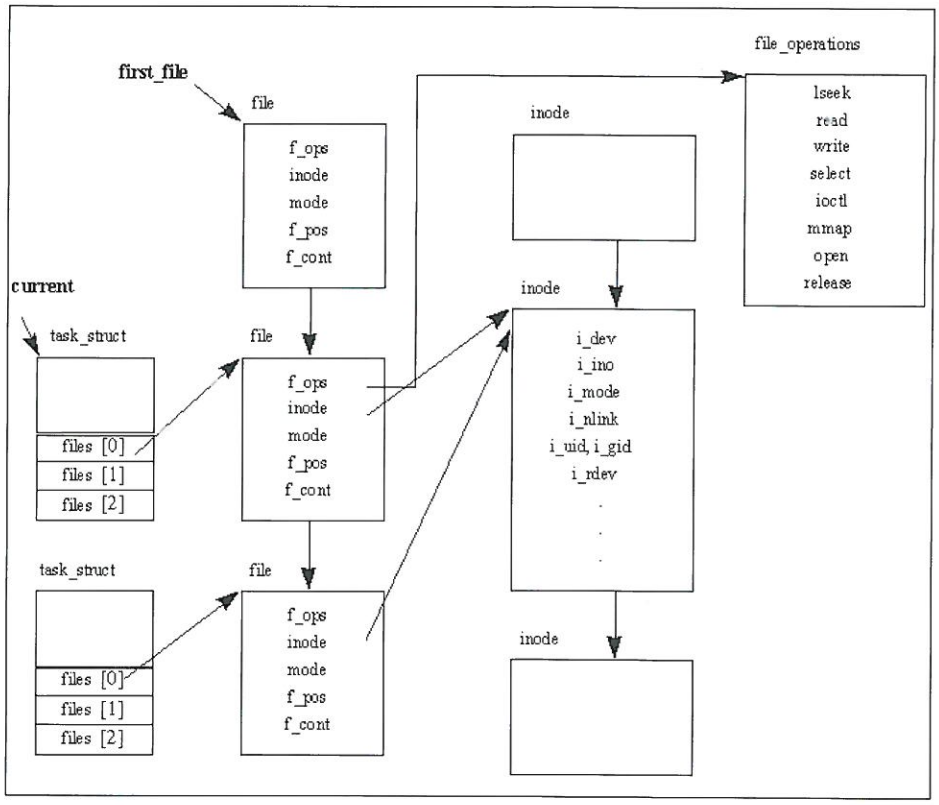


Fig. 4 Estructuras de datos internas para la gestión de la E/S.

```
/* copiar.c */
#include <sys/file.h>
#include <limits.h>

main(int argc, char * argv[])
{
int source_fd, target_fd;
char c;

if((source_fd=open(argv[1],O_RDONLY))<0)
        perror("copiar");
if((target_fd=creat(argv[2],0666))<0)
        perror("copiar");

while(read(source_fd,&c,1)>0)
      write(target_fd,&c,1);

close(source_fd);
close(target_fd);
}
```

```
$ copiar f1 f2
        Copia el contenido del fichero f1 en el
        fichero f2.

$ copiar f1 /dev/tty
        Copia el contenido del fichero f1 en el
        terminal. El resultador obtenido es equi-
        valente a la ejecución de cat f1.
```

Fig. 3 El programa copiar.c puede operar con ficheros y dispositivos.

ENLACE ENTRE LA OPERACIÓN UNIFORME Y LA OPERACIÓN ESPECÍFICA

Cuando un proceso ejecuta una operación uniforme (*read* por ejemplo), transfiere el control a una rutina del kernel denominada *syscall*. Ésta, recibe un parámetro que identifica la operación ejecutada (*open*, *close*, *read*, *write*, ...). En función de este parámetro, *syscall* salta a la rutina genérica asociada, (*sys_read* en el ejemplo) que recibe los

parámetros especificados por el usuario en la llamada a la operación uniforme.

Esta rutina, utiliza el fd para indexar en la FDT, localizar la estructura file asociada, y a partir de ésta hallar la dirección de la rutina *read_dev* implicada, dentro de la estructura *file_operations*. Una vez localizada, la rutina es ejecutada pasándole como parámetros: el inode, la estructura file del fichero/dispositivo afectado, la dirección de memoria donde debe dejarse la información leída y el número de bytes a leer. La rutina específica (*read_dev*) es la responsable de realizar la lectura física desde el dispositivo, y situar la información en la zona de memoria donde el proceso espera encontrarla. La Fig. 5 muestra la secuencia de operaciones que tienen lugar desde que un proceso de usuario ejecuta un operación uniforme, hasta que la rutina específica gana el control.

DRIVERS DE CARACTERES

Un driver de caracteres está formado por el conjunto de rutinas específicas del dispositivo, la estructura file_operations (ver Fig.6), más una rutina interna de inicialización. Se analiza a continuación las rutinas que forman parte del driver.

- *long init_char_dev(long memstart)*

Punto de entrada: Durante el proceso de boot. Es necesario incluir una llamada a esta rutina en el fichero /usr/src/linux/drivers/char/mem.c

Función: Inicialización de las variables internas del driver y reserva dinámica de memoria si es necesario. Debe registrar el driver dentro de las estructuras de datos internas del kernel, mediante la rutina *register_char_dev()* (ver Fig. 7).

Parámetros: *memstart* contiene la última dirección de memoria utilizada, a partir de la cual el driver puede reservar el espacio que necesite y retornar la nueva dirección final de memoria.

Retorno: dirección final de memoria después de inicializar el driver.

- *int open_char_dev(struct inode *inode, struct file *file)*

Punto de entrada: Durante una operación open de un dispositivo.

Función: Realiza las inicializaciones y controles asociados a la apertura de un dispositivo físico. Por ejemplo, para un dispositivo no compartible, debería asegurar que sólo un proceso lo mantenga abierto simultáneamente.

Parámetros:

inode inode del dispositivo.

file elemento de la LFA asociado al open.

(definidos en /usr/include/linux/fs.h).

Retorno:

- *EBUSY* dispositivo no compartible ocupado.
- *ENODEV* dispositivo inexistente.

- *void release_char_dev(struct inode * inode, struct file *file)*

Punto de entrada: Asociado a la operación *close()*.

Función: Realiza las operaciones relativas al cierre de un dispositivo físico. Por ejemplo, marcar el dispositivo como libre, liberar memoria dinámica asignada en el proceso de open, etc ...

Parámetros: ver *open_char_dev()*.

Retorno: No definido.

- *int read_char_dev(struct inode * inode, struct file * file, char * buff, int count)*

Punto de entrada: Es llamada cada vez que se ejecuta una operación uniforme *read*.

Función: Accede al dispositivo, lee el número de bytes indicados en la variable *count*, y los sitúa en un buffer interno del driver. Esta información debe posteriormente ser copiada en el espació de memoria de usuario utilizando la rutina *memcpy_tofs(user_addr,system_addr,n)*. Esta copia no puede realizarse con la rutina memcpy standard de C, debido a que el espacio de memoria del kernel (donde se halla el driver) y el espacio de memoria de usuario son disjuntos.

```
struct file_operations {
int (*lseek) (struct inode *, struct file *, off_t,
int);
int (*read) (struct inode *, struct file *, char *,
int);
int (*write) (struct inode *, struct file *, char *,
int);
int (*readdir) (struct inode *, struct file *, struct
dirent *, int);
int (*select) (struct inode *, struct file *, int,
select_table *);
int (*ioctl) (struct inode *, struct file *, unsigned
int, unsigned long);
int (*mmap) (struct inode *, struct file *, struct
vm_area_struct *);
int (*open) (struct inode *, struct file *);
void (*release) (struct inode *, struct file *);
int (*fsync) (struct inode *, struct file *);
int (*fasync) (struct inode *, struct file *, int);
int (*check_media_change) (dev_t dev);
int (*revalidate) (dev_t dev);
};
```

Fig. 6 Estructura file_operations. El resto de rutinas son utilizadas por Linux para implementar el VFS (Virtual Filesystem Switch).

Esta rutina es responsable, también, de actualizar el puntero de lectura/escritura.

Parámetros:

inode, file

buff dirección de usuario donde debe dejarse la información leída.

count número de bytes a leer.

Retorno:

- **EINVAL** cuando la rutina read_char_dev no está accesible desde la estructura file_operations. Por ejemplo, dispositivo de solo salida.

>0 número de bytes leídos realmente (si la operación ha tenido éxito).

-1 cuando se produce un error.

- *int write_char_dev(struct inode * inode, struct file * file, char * buff, int count)*

Punto de entrada: Es llamada cada vez que se ejecuta una operación uniforme write.

Función: Copia en un buffer del driver el número de bytes indicados por count, situados a partir de la dirección de usuario buff. Esta copia debe realizarse, por los motivos expuestos anteriormente, utilizando la función memcpy_fromfs(system_addr,user_add

Linux ofrece un conjunto uniforme de operaciones para operar con ficheros y dispositivos

r,n). Posteriormente escribe, sobre el dispositivo, la información del buffer y actualiza el puntero de lectura/escritura.

Parámetros: ver *read_char_dev()*.

Retorno:

- **EINVAL** cuando la rutina write_char_dev no está accesible desde la estructura file_operations.

>0 número de bytes escritos realmente (si la operación ha tenido éxito).

-1 cuando se produce un error.

- ***int ioctl_char_dev(struct inode * inode, struct file *file, unsigned int cmd, unsigned long arg)***

Punto de entrada: Es llamada cada vez que se ejecuta la operación uniforme *ioctl*.

Función: Permite implementar todas aquellas operaciones específicas a un dispositivo que no pueden englobarse dentro de las operaciones read y write.

Parámetros:

inode, file

cmd codigo de comando. Identifica la operación que el usuario solicita del driver. El significado de este valor debe ser establecido por el programador del driver.

arg es un long, y por tanto, tiene la misma longitud en bytes que un puntero. Un usuario puede entonces comunicar los parámetros asociados a la operación cmd, pasando a través de arg un puntero a una estructura que los contenga.

Retorno:

- **EINVAL** cuando la rutina *ioctl_char_dev* no está accesible desde la estructura *file_operations*.

n definido por el programador.

- ***int lseek(struct inode *, struct file * file, off_t offset, int origin)***

Punto de entrada: Cada vez que se ejecuta la llamada al sistema *lseek*.

Función: Posicionar el puntero de lectura/escritura. Posicionar de forma conveniente el dispositivo físico, en caso de ser necesario (Por ejemplo, rebobinar una cinta). Si no se especifica esta rutina, por defecto el sistema actualizará la variable f_pos (valor del puntero) de la estructura file asociada.

Parámetros:

inode, file

offset indica el nuevo valor del puntero de lectura/escritura. Debe ser interpretado según el valor de origin.

Origin; **0** posicionamiento absoluto, **1** posicionamiento relativo a la posición actual, **2** posicionamiento relativo al final del fichero.

retorno:

- **EINVAL** cuando la rutina *lseek_char_dev* no está accesible desde la estructura file_operations.

>0 nueva posición absoluta del puntero.

<0 error definido por el programador.

Para cada operación uniforme, existe una rutina específica dentro del driver del dispositivo

MAILBOX: UN EJEMPLO.

Un mailbox es un dispositivo lógico compartible, que actúa de forma similar a los ficheros fifo de UNIX y que se utiliza básicamente como mecanismo de comunicación entre procesos (I.P.C.).

Un proceso puede abrir un mailbox y escribir información en él. Esta información quedará almacenada hasta que otro proceso lo abra y la lea. En este momento, la información leída desaparecerá, por lo tanto leer equivale a consumir. Al ser un dispositivo compartible permite establecer un esquema de comunicación entre m procesos escritores y n procesos lectores.

Las operaciones definidas sobre un mailbox son las siguientes:

- **lectura *(read):*** Las lecturas son atómicas. Si un proceso intenta leer 100 caracteres de un mailbox, y en este solo hay 75 caracteres almacenados, la operación no finaliza hasta que se puedan leer los 25 caracteres restantes. Para ello, algún otro proceso deberá escribirlos. Una operación de lectura sobre un mailbox vacío provoca por lo tanto:
  - El bloqueo del proceso lector en el caso que no haya existido ningún proceso escritor.

```
if
(register_char_dev(MAJOR_NR,"mbx",&mbx_fops)
)
        {
        /* ERROR: ya hay otro dispositivo utili-
zando este mayor */
        printk("mbx DRIVER: Error imposible
conseguir major %d\n",MAJOR_NR);
        return(0);
        }
```

Fig. 7 Registro del driver de caracteres.

  - La recepción de un EOF en el caso que no exista ningún proceso escritor, pero haya existido previamente.
- **escritura *(write):*** No existen operaciones de escritura parciales. Si al iniciar una operación de escritura se detecta que no hay bastante espacio libre dentro del mailbox, como para almacenar toda la información que se quiere escribir, el proceso productor quedará bloqueado hasta que haya suficiente espacio libre. Un proceso que escribe sobre un

```
typedef struct smbx
    {
    char buf[TAMAÑO];
    int in;/* indice al siguiente elemento a rellenar
*/
    int out; /* indice al siguiente elemento a leer */
    int nCar; /* número de caracteres ocupados */
    struct wait_queue * lleno; /* sobre este evento,
se duerme un proceso que escribe en un mbx
lleno */
    struct wait_queue * vacio; /* sobre este evento,
se duerme un proceso que lee un mbx vacio */
    int dormidoVacio; /*contabiliza el nº de proce-
sos dormidos sobre el evento vacio*/
    int dormidoLleno; /* contabiliza el nº de proce-
sos dormidos sobre el evento lleno*/
    int nLectores; /*contabiliza el nº de procesos
que leen del mbx */
    int nEscritores;  /*contabiliza el nº de procesos
que escriben  en el mbx */
    BOOLEAN BloqLect; /* TRUE cuando una ope-
ración de lectura sobre un mbx
    vacío y sin procesos escritores, debe bloquear
al lector */;
    BOOLEAN EnviarSignal; /* TRUE cuando una
operación de escritura sobre un mbx
    sin lectores, provoca  un signal SIGPIPE al
escritor */
    } SMBX;
```

Fig. 8 Estructura de datos del mailbox.

FIGURA 9

```
int mbx_init(long mem_start)
                {

register_chrdev(MAJOR_NR,"mbx",&mbx_fops);
/* registrar mailbox */
                for (i=0; i<MAX_MINOR;i++)
                        {
                        mbx[i].in=0;
                        mbx[i].out=0;
                        mbx[i].nCar=0;

mbx[i].dormidoVacio=mbx[i].dormidoLleno=0;
        mbx[i].nLectores=mbx[i].nEscritores=0;
        mbx[i].BloqLect=TRUE;
        mbx[i].EnviarSignal=FALSE;
        }
        }
int mbx_open(struct inode * inode, struct file
*file)
        {
        minor=MINOR(inode->rdev); /* minor =
minor del dispositivo */
        if (modo de apertura == O_RDONLY)
mbx[minor].nLectores++;
        else
        if (modo de apertura == O_WRONLY)
mbx[minor].nEscritores++;
        else
        if (modo de apertura == O_RDWR)
        {
        mbx[minor].nEscritores++;
        mbx[minor].nLectores++;
        }
}
void mbx_release(struct inode *inode, struct
file *file)
        {
        minor=MINOR(inode->rdev); /* minor =
minor del dispositivo */
        if (modo de apertura == O_RDONLY)
mbx[minor].nLectores--;
        else
        if (modo de apertura == O_WRONLY)
mbx[minor].nEscritores--;

        else
        if (modo de apertura == O_RDWR)
        {
        mbx[minor].nLectores--;
        mbx[minor].nEscritores--;
        }

mbx[minor].BloqLect=((mbx[minor].nLectores==0)
&& (mbx[minor].nEscritores==0));

mbx[minor].EnviarSignal=!((mbx[minor].nLectores=
=0) && (mbx[minor].nEscritores==0));
        }
        int mbx_read(struct inode * inode,
struct file * file, char * buf, int count)
        {
        minor=MINOR(inode->rdev); /* minor =
minor del dispositivo */
        while (i<count)
        {
        if (buffer vacío)
```

```
        if (mbx[minor].BloqLect)
esperar hasta buffer no vacío;
else    return 0; /* retorna EOF */
copiar un carácter del buffer del mailbox al buffer
del proceso;
mbx[minor].out=(mbx[minor].out+1) % TAMAÑO;
mbx[minor].nCar--;
if (hay procesos esperando que el buffer no este
lleno) despertarlos;
i++;
}
return(i);  /* retorna el nº de caracteres leidos */
}
int mbx_write(struct inode * inode, struct file *
file, char * buf, int count)
{
minor=MINOR(inode->rdev); /* minor = minor del
dispositivo */
if ((count>0) && (mbx[minor].nLectores==0) &&
(mbx[minor].EnviarSignal))
{
enviar signal al proceso actual;
return -1;
}
if (buffer lleno) esperar hasta buffer no lleno;
while (i<count)
{
copiar un carácter del buffer del proceso al buffer
del mailbox;
mbx[minor].in=(mbx[minor].in+1) % TAMAÑO;
mbx[minor].nCar++;
if (hay procesos dormidos debido a buffer vacío)
despertarlos;
i++;
}
return (i); /* retorna el nº de caracteres escritos */
}
```

mailbox que no tiene ningún proceso lector, pero lo ha tenido previamente, recibe un signal *SIGPIPE*.

- **depuración (debug):** Esta operación permite consultar las variables internas del driver. Tiene utilidad de debug y está accesible a través de la llamada al sistema *ioctl* (cmd=1).
- **inicialización:** Inicializa las estructuras de datos asociadas a un dispositivo. Tiene utilidad de debug, ya que durante el proceso de boot ya se produce la inicialización del driver. Esta operación está accesible a través de la llamada al sistema *ioctl* (cmd=2).

IMPLEMENTACIÓN.

Se analizan a continuación las estructuras de datos necesarias para implementar el driver (ver fig.8).

Un mailbox consta de un buffer circular de tamaño finito (4Kb en la implementación propuesta) ubicado en memoria y gestionado por dos indices: *in*, índice que apunta al primer elemento a rellenar y *out*, índice que apunta al primer elemento a consumir. Se dispone, además, de un evento sobre el cual, dormir los procesos que escriben sobre un mailbox *lleno* y otro sobre el cual, dormir los procesos que leen de un mailbox *vacio*. La variable *dormidosVacio* contabiliza el número de procesos dormidos sobre el evento *vacio* y la variable *dormidosLleno* contabiliza los dormidos sobre el evento *lleno*. El driver define *MAX_MINOR* (5 en la implementación propuesta) mailboxes sobre la tabla *mbx*, lo cual significa que podremos crear 5 ficheros de dispositivo tipo mbx, con minors 0..4.

La Fig.9 muestra, en un pseudocódigo estilo C, la implementación de dichas rutinas. Se han obviado en éste, algunas características específicas de la implementación, para simplificar su comprensión. El código fuente real se encuentra en el disco que acompaña la revista.

EJEMPLO DE UTILIZACIÓN

Se puede comprobar fácilmente el funcionamiento del driver mediante el siguiente procedimiento:

1. Abrir dos ventanas de terminal.
2. Ejecutar, en la primera ventana, la siguiente orden:
   $ more mbx0
3. Ejecutar, en la segunda ventana, la siguiente orden:
   $ ls -lR / > mbx0

Como resultado se observará que la salida de la orden ls -lR aparecerá por la primera ventana. La secuencia de ordenes anterior es equivalente a la ejecución de:

$ ls -lR / | more

Contactar con los autores:
E-mail: pere@ei.udg

BIBLIOGRAFÍA

[Johnson93] Linux Kernel Hacker's Guide. Michael K. Johnson. 1993. johnson@sunsite.-unc.edu

[Pajari92] Writing Unix Device Drivers. George Pajari. Addison Wesley. 1992.

[Leffler89] The design and Implementation of the Unix Operating System. Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, Jon S. Quaterman. Addison wesley. 1989.

[Bach86] The Design of the Unix Operating System. Maurice J. Bah. Prentice Hall international. 1986.

CONSIDERACIONES FINALES

Juan Manuel y Luis Martín

Con el presente artículo termina la serie dedicada a la programación orientada al manejo de Bases de Datos con el lenguaje Clipper. En este último artículo se pretende dar una serie de consideraciones finales, así como algunos consejos prácticos que puedan resultar de utilidad al programador a la hora de crear sus aplicaciones.

Como se ha venido explicando a lo largo de los anteriores artículos, Clipper es un lenguaje básicamente orientado al manejo de bases de datos. Esta característica no implica que Clipper no pueda ser utilizado para la creación de aplicaciones con otros propósitos, ya que, a pesar de nacer como una versión compilada del lenguaje dBASE, ha evolucionado como un lenguaje de programación independiente. De esta forma, el programador encontrará en este lenguaje las mismas características que en cualquier otro lenguaje de programación de propósito general (estructuras de control estándar, diversos tipos de datos, subrutinas, etc.)

BASES DE DATOS

Clipper está especialmente diseñado para la creación de aplicaciones orientadas al manejo de bases de datos en entornos de PC's y Redes de Área Local. La utilización de bases de datos en las aplicaciones creadas con Clipper es realmente potente y, a la vez, su manejo resulta muy sencillo.

A este respecto, la parte que suele entrañar una mayor dificultad es la concepción general de la aplicación. Así, un buen diseño previo puede resultar fundamental tanto para el correcto funcionamiento de la aplicación, como para la codificación y depuración de la misma.

Otro de los aspectos importantes en el trabajo con bases de datos es el relacionado con la integridad de la información almacenada. Una de las máximas que resulta conveniente seguir consiste en no almacenar la misma información en dos sitios a la vez. Si en determinado momento se actualiza la información sólo en uno de ellos, puede producirse un grave problema de inconsistencia a la hora de acceder a la misma. Este tipo de errores es muy común, y si no se controla, puede llegarse a un punto en que resulte imposible saber cual es la información correcta y cual la errónea.

Cuando se va a trabajar con varias bases de datos al mismo tiempo resulta siempre conveniente abrirlas todas al principio de la aplicación, cada una de ellas en un área de trabajo. De esta forma, el código resultará mucho mejor organizado, aunque el consumo de memoria sea sensiblemente superior. Además, también resulta conveniente dotar a las áreas de trabajo de su correspondiente alias, con el fin de poder identificar de forma clara y concisa la base de datos sobre la que se está trabajando.

Otra consideración importante relacionada con el manejo de bases de datos es la utilización del operador alias (->) a la hora de acceder a los distintos campos. Este operador garantiza que el acceso se realiza a los campos de una base de datos en concreto, y no a los del área de trabajo actual, evitando los conflictos que puedan surgir por la utilización de nombres de campo idénticos.

El trabajo con los índices también puede ser el causante de problemas si no se realiza con cautela. Debe tenerse siempre presente que los índices sólo se actualizan automáticamente cuando se encuentran abiertos, independientemente de cual de ellos éste activo. La corrupción de los índices puede ser origen de problemas que ocasionen la obtención de datos incorrectos.

A la hora de reconstruir un índice debe también tenerse presente si éste es o no condicional, es decir, si incluía cláusulas de ámbito, *WHILE, FOR* y *ASCENDING* o *DESCENDING* en el momento de sus creación. En este caso,

hay que recordar que en el fichero de índice se almacenan únicamente las cláusulas *FOR*, *ASCENDING* y *DESCENDING*, mientras que las cláusulas de ámbito y *WHILE* se pierden. Por ello, a la hora de realizar la reconstrucción del índice mediante el mandato *REINDEX*, solamente tendrán efecto las primeras.

PROGRAMACIÓN ORIENTADA A OBJETOS

Como ya se indicó en anteriores artículos, Clipper no puede ser considerado un lenguaje orientado a objetos, sino más bien un lenguaje estructurado y procedimental con una aproximación a la OOP. La principal limitación del lenguaje en este sentido es la imposibilidad del programador tanto para crear nuevas clases como para alterar las ya existentes.

A pesar de ello, será rara la vez que el programador necesite ampliar las clases existentes, ya que estas son realmente completas y generalmente suele utilizarse toda su funcionalidad al completo.

Como se indicó en el artículo correspondiente a la nueva versión 5.3, el número de clases ha sido ampliado, dotando al lenguaje de una colección de clases para la definición de controles que agradarán tanto al programador como al usuario. De cualquier forma, se sigue echando en falta la entrada completa de este lenguaje en el mundo de la OOP.

LA CLASE GET

Uno de las herramientas mas potentes que este lenguaje pone a disposición del programador es la clase Get. Se trata de una clase que permite crear objetos para realizar la edición tanto de campos de bases de datos como de variables de memoria. Su principal ventaja es que se trata de una clase muy estructurada, permitiendo trabajar al programador con diferentes niveles de control sobre el proceso de edición.

El nivel más básico de funcionamiento es el denominado nivel *Read*. A este nivel, el proceso de edición se realiza mediante los tradicionales mandatos *@...GET y READ*. Siempre resulta recomendable utilizar este nivel si con él pueden realizarse las operaciones que el programador necesita, ya que, en este nivel la manipulación de los objetos es realizada el propio sistema de forma automática.

El siguiente nivel es el denominado nivel *Reader*. A este nivel, el proceso de edición realizado por cada uno de los objetos Get debe ser especificado por separado, permitiendo diseñar un tipo de edición independiente para cada uno de ellos. Este nivel incrementa ligeramente la complejidad en la manipulación de los objetos, ya que es el propio programador el encargado de definir las acciones a realizar con cada uno de ellos.

Por último el nivel mas bajo del sistema Get es nivel de Funciones. Este es el nivel que otorga un mayor control al programador sobre el proceso de edición. Como contrapartida, la complejidad en el manejo de los objetos aumenta de forma considerable. Por tanto, su utilización sólo es recomendable en aquellos casos en sea estrictamente necesario.

LA CLASE TBROWSE

Otra de las herramientas surgidas de la aproximación de Clipper al mundo de la Programación Orientada a Objetos son las tablas deslizantes. Se trata de objetos derivados de la clase TBrowse, que permiten visualizar la información almacenada en bases de datos y matrices en forma de tabla bidimensional. Esta forma de presentar la información resulta muy elegante, a la vez que cómoda de utilizar por parte del usuario.

La inclusión de esta clase en el lenguaje, además de facilitar la tarea de presentación de la información en la pantalla, ofrece tal versatilidad que, utilizada en conjunción con la clase Get, permite realizar prácticamente cualquier operación. La clase TBrowse es posiblemente la que más diversas utilizaciones puede tener dentro de una aplicación. Sirvan como ejemplo las siguientes sugerencias:

- Edición de archivos de texto, utilizando las filas del objeto Browse como las líneas del texto.
- Realización de menús desplegables tipo *ACHOICE()*, codificando las distintas opciones en cada una de las filas del objeto.
- Inclusión de ayudas en línea mediante, utilizando la fila y columna en la que se encuentra el cursor para visualizar la ayuda correspondiente.

PROGRAMACIÓN EN RED

Otro de los aspectos que tiene gran importancia dentro de la creación de aplicaciones encaminadas a la manipulación de bases de datos es la posibilidad de contar con herramientas para el trabajo en un entorno de red. Clipper proporciona herramientas para el trabajo en red que el programador puede utilizar de una forma muy sencilla.

El método utilizado por Clipper para el trabajo en red es el sistema de bloqueos. Así, se ofrece la posibilidad de bloquear por parte del usuario tanto una base de datos entera como un único registro de la misma. De esta forma, se garantiza que los accesos a la información de la base de datos no generarán conflictos con el resto de los usuarios.

Para aquellos programadores cuyo trabajo se desarrolle dentro de un entorno de red, Clipper puede llegar a ser la herramienta perfecta. Sin embargo, Clipper proporciona la capacidad de bloquear por completo las bases de datos para aquellas operaciones que requieren un grado de exclusividad total (reindexación de ficheros, empaquetado de la información, etc). No obstante, el abuso de la exclusividad en el bloqueo puede redundar en una falta de eficiencia de la red, ya que se niega el acceso a los datos al resto de los usuarios. Por ello, los bloqueos deben utilizarse siempre de una forma racional, es decir, utilizando siempre un nivel de bloqueo suficiente como para garantizar la integridad de los datos, pero que al mismo tiempo permita el mayor acceso posible del resto de usuarios de la red.

LOS RDD's

Otra de las grandes novedades incluida en las últimas versiones del lenguaje es la posibilidad de utilizar los denominados Controladores Sustituibles de Bases de Datos o *Replaceable Database Drivers* (RDD). Mediante la correcta utilización de estos controladores es posible manipular bases de datos almacenadas en dife-

rentes formatos, usando para ello las mismas funciones y mandatos que para el formato estándar.

Este tipo de arquitectura de funcionamiento fue implantado a partir de la versión 5.0, pero es a partir de la versión 5.2 cuando empieza a ser realmente operativa. Para utilizar una base de datos en un formato distinto al estándar de Clipper, basta con indicar el controlador correspondiente en el momento de su apertura o creación.

La arquitectura basada en la utilización de controladores de bases de datos sustituibles proporciona al programador un nivel de abstracción que posibilita el trabajo con cualquiera de los formatos de bases de datos soportados sin necesidad de aprender nuevas funciones ni mandatos para cada uno de ellos.

Otro de los aspectos importantes en el trabajo con bases de datos es el relacionado con la integridad de la información

GESTIÓN DE ERRORES

Clipper también proporciona un potente sistema para la gestión de errores en tiempo de ejecución. Se trata de un sistema bastante abierto, ya que permite al programador alterar su forma de funcionamiento para adecuarlo a las circunstancias que concurran en cada caso. De esta forma, el programador puede definir distintos tipos de actuación en función tanto del error como del estado interno de la propia aplicación.

Este sistema está basado en una clase denominada *Error*. Se trata de una clase de la que se derivan objetos que únicamente posee atributos. En estos atributos es donde el sistema de errores de Clipper almacena la información relativa al error que se ha producido. Una vez cargados los valores correspondientes, el programador puede consultarlos y realizar las operaciones oportunas.

Existen además dos formas de controlar las operaciones a realizar en caso de error. La primera de ellas consiste en la utilización de la estructura SEQUENCE. Se trata de una estructura similar a las estructuras de control estándar, con la diferencia de que, en ella, es posible incluir una serie de operaciones que únicamente serán ejecutadas en caso de que se produzca un error.

La segunda forma consiste en la definición de bloques de código personalizados para el tratamiento de los diferentes tipos de error. Esto es posible gracias a que Clipper define las operaciones por defecto que deben realizarse en caso de error mediante un bloque de código. Utilizando la función ERRORBLOCK() es posible cambiar el bloque de código que será evaluado al producirse el error. De esta forma, el programador puede definir sus propios bloques de código personalizados para los distintos tipos de gestión que deben realizarse. El bloque de código para el tratamiento de errores puede ser sustituido cuantas veces se desee a lo largo del programa, permitiendo así realizar muy distintas operaciones de tratamiento para cada una de las partes que componen el código de una aplicación.

OTROS ASPECTOS IMPORTANTES

Como el lector ha podido comprobar a lo largo de esta serie de artículos, Clipper es quizá el lenguaje de programación orientado al manejo de bases de datos más potente y completo del mercado, dentro del entorno de la microinformática. Además de los aspectos y herramientas explicados en los distintos artículos, Clipper posee otras herramientas de carácter muy avanzado que hacen de este lenguaje uno de los mas versátiles, a la vez que potente y sencillo de manejar.

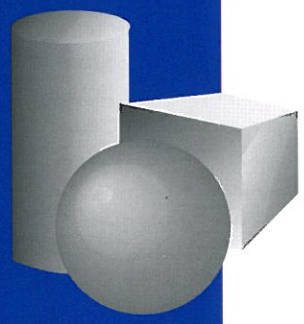
Entre las herramientas mas avanzadas de las que el programador puede hacer uso, caben destacar las siguientes:

- Enlace Clipper-C: Debido a que el propio Clipper está desarrollado en lenguaje C, la forma de interactuar entre ambos lenguajes es realmente potente. Así, resulta realmente sencilla la inclusión de funciones desarrolladas y compiladas en C dentro de una aplicación creada con Clipper. De hecho, la llamada se realiza directamente, y sólo será necesario enlazar juntos los módulos objeto de Clipper y de C. Mediante al inclusión de funciones desarrolladas en lenguaje C es posible dotar a Clipper de todas las herramientas que posee el primero, permitiéndole utilizar funciones gráficas, acceso a los recursos hardware, etc.
- Gestión de Archivos a Bajo Nivel: Clipper dispone de un amplio repertorio de funciones que permiten al programador acceder a los ficheros a bajo nivel, es decir, acceder directamente a los bytes que lo componen. De esta forma, es posible crear programas orientados también al tratamiento de ficheros de texto, ficheros binarios, directorios, etc.
- Librerías Preenlazadas: Clipper permite utilizar librerías cuyo código se encuentra enlazado en un fichero separado del ejecutable. Se trata de ficheros con la extensión PLL (*Prelinked Libraries*), que deben siempre acompañar al fichero ejecutable, pero que, como contrapartida, ofrecen la posibilidad de ser compartidos por otras aplicaciones. La utilización de este tipo de librerías es especialmente recomendable en aplicaciones constituidas por múltiples ficheros ejecuatables que comparten fragmentos de código. Su utilización redunda en un considerable ahorro de espacio en disco, así como en una importante reducción de tiempo en los procesos de compilación y enlace.

CONCLUSIÓN

La misión principal de esta serie de artículos ha sido introducir al lector en una de las herramientas de trabajo más utilizadas dentro del ámbito laboral, así como mostrar el funcionamiento de algunas de sus principales herramientas.

Como punto final, es importante recordar al lector que el perfecto conocimiento y dominio de un lenguaje de programación se alcanza principalmente mediante la práctica. Todas las explicaciones dadas a lo largo de estos artículos no sirven de nada si el programador no se sitúa delante del ordenador y realiza tanto los ejemplos propuestos como los sus suyos propios.



JUEGOS CONVERSACIONALES

Ignacio Cea

A los largo de todas estas semanas hemos tratado de desglosar cuáles son las líneas generales que deberían seguirse para diseñar una librería de clases para la construcción de juegos conversacionales. En esta última entrega trataremos de cortar los hilos pendientes ocupándonos del gobierno general de un juego

Un juego conversacional no es nada sin alguien que gobierne la interacción de todos sus personajes y objetos. El objetivo de este artículo en esbozar las líneas generales del diseño de ese alguien a la par que cerrar los hilos pendientes de anteriores artículos.

ALEATORIEDAD = JUEGO CONVERSACIONAL

Si hay algún factor que es imprescindible tener en cuenta a la hora de diseñar un juego conversacional o, en nuestro caso, una librería de clases para construirlos, es la aleatoriedad. No es posible concebir un juego, medianamente entretenido en el que algunas cosas no tengan un comportamiento diferente entre ejecuciones distintas.

Dicho factor está aquí presente en la forma de dioses y catástrofes naturales. Los dioses, en momentos claramente aleatorios, actúan sobre el mundo imaginario donde se desarrolla el juego, provocando sobre el una catástrofe natural de consecuencias insospechadas.

Este artículo hace hincapié, sobre todo, en el diseño de esta entidad y su ubicación dentro del sistema.

UN JUEGO POLITEÍSTA

Dios es la entidad encargada de introducir la aleatoriedad de hechos dentro de un juego conversacional. "Dios" es una clase abstracta (tiene al menos un método virtual puro) en la que se declara un método llamado "actuar". Ese método especifica la forma en la que el dios afecta al mundo. Debe ser redefinido dentro de las clases hijas.

Por tanto, no estamos considerando un mundo monoteísta sino más bien todo lo contrario. Nuestros dioses son entes sumamente especializados que sólo conocen una forma de relacionarse con el mundo.

Parece en todo momento que el dios que estamos considerando en este diseño es del todo malvado, porque la única forma que tiene de actuar son las catástrofes naturales. En realidad la entidad dios en alguien que tiene una forma de actuar y esa puede ser o bien una catástrofe natural o bien un milagro. Son las propias clases hijas de dios quien deben especificar eso aunque aquí las clases que se mencionan sean siempre catástrofes.

EL GOBERNADOR

El nombre de "Dios", sugiere siempre una entidad independiente que, a su libre albedrío, decide interactuar con el mundo de una forma cualquiera. Ya hemos visto en el anterior apartado que esa forma no es cualquiera sino una concreta definida en cada clase hija. Sin embargo, el concepto de independiente sigue estando presente.

Independencia, llevada a la máquina supone considerar, cuando menos, sistemas operativos multitarea (UNIX y OS/2). Lo ideal sería que dentro del mismo ejecutable, los dioses fueran "clases" completamente independiente del hilo general; es decir que, aun dentro del mismo ejecutables siguieran hilos independientes. Esa posibilidad sólo está disponible en los sistema operativos multihilo (OS/2 y algunas versiones recientes de UNIX).

Sin embargo, el diseño de la librería conversacional está, en todo momento, siendo pensada para sistemas operativos monotarea (MSDOS, Windows). Por tanto, el control de cuando actúan o no debe estar situado en la clase que se ocupa del gobierno general del juego;esto es, en: "Juego

Conversacional" y cuya forma de funcionar sea, básicamente, el *pooling*. Juego conversacional es, por tanto, el gobernados de la librería.

PRIMERA CONCLUSIÓN

Tenemos, por tanto, dos importantes clases: "Juego Conversacional" y "Dios". La primera gobierna el juego y decide, aleatoriamente, cuando deben o no actuar las instancias de las otras clases sobre el mundo.

En las próximas líneas nos vamos a centrar en esbozar los aspectos generales de ambas.

UN DISEÑO MALO DE DIOS

El aspecto básico de la clase Dios y una hija suya, podría ser el siguiente:

```
class Dios
{

public :

Dios (const String&);
~Dios ();
const String& nombre (void) const;
virtual ResultadoAccion actuar (void) = 0;

protected :

String elNombre;

};

class Jupiter : public Dios
        {

public :

Jupiter (const String&);
ResultadoAccion actuar (void);

};
```

Para encontrar fallos a estas clases o, más bien, a su diseño, hay que fijarse detenidamente en su ubicación dentro del resto de la librería. De un lado, los dioses son únicos. Nadie impide, por ejemplo, tal y como está escrita la clase Jupiter que exista más de una instancia de ella. Digamos que el único control estaría por el uso que de ella hiciera el programador del juego conversacional. De otro, los dioses deben estar todos referenciados para poder acceder a ellos, lo cual tiene un grave inconveniente: el código para elegir aleatoriamente a uno de ellos debe estar lleno de instrucciones switch y case lo que impide que un usuario cualquiera pueda ampliar la librería sin tocar esa instrucción switch que, por otro lado estará dentro de un fichero .lib innaccesible.

IDENTIFICANDO A LOS DIOSES

Hagamos uso del mismo concepto empleado para registrar las distintas acciones (ver capítulo anterior). Vamos a construir una agencia de dioses. Cada vez que se cree uno de ellos, lo registraremos dentro de una lista estática dentro de la clase Dios y colocaremos un método para poder buscar uno dentro de ella aleatoriamente. La clase quedaría, básicamente:

```
class Dios
{

public :

Dios (const String& n) : nombre (n)
{ Dios::losDioses.insertar (this); }
static Dios* unDios (void);
...

protected :

...
static Lista<Dios*> losDioses;

};
```

LA UNIDAD DE LOS DIOSES

Hasta este momento nada que no haya visto ya. Sin embargo tanto ahora, como en el mencionado caso de las acciones necesitamos garantizar su unicidad. ¿Cómo podría hacerse esto?. Simple, cada vez que creemos un objeto dios (o uno acción) buscaremos, antes de insertarlo en la lista otro que se denomine igual. Si eso es así abortamos la construcción sin insertar el nuevo elemento dentro

```
class Dios
{

public :

Dios (const String& n) : nombre (n)
{
if (Dios::dios (elNombre) != NULL)
return;
else Dios::losDioses.insertar (this);
}
static Dios* unDios (void);
static Dios* dios (const String&);
...

protected :

...
static Lista<Dios*> losDioses;

}
```

de la lista por lo que, a todos los efectos, es como si no existiera. La clase quedaría:

ARREGLAR UN PEQUEÑO FALLO

Sin embargo, nos volvemos a quedar a un paso del resultado final. La parte de la librería que se encargue de crear los dioses, nunca podrá saber si el que acaba de construir ha sido o no insertado dentro de la lista y, por tanto, si lo que debe hacer con él (borrarlo, cambiarle el nombre, etc...). Necesitamos una manera de poder indicar que el proceso de construcción ha fallado. Si se tratara de cualquier otro método, con devolver un código de error estaría solucionado, pero estamos hablando del constructor que no puede retornar nada.

Los modernos C++ disponen del mecanismo de excepciones tan común en otros lenguajes (instrucciones *throw*, *try* y *catch*). Sin embargo, al no ser algo comúnmente extendido aún, vamos a hacerlo de una forma más pedestre.

Vamos a colocar una variable interna de implementación dentro de los objetos dioses que nos indiquen si ha sido o no construido con exactitud. Tras finalizar el proceso de construcción no tenemos más que consultar

esa variable y tomar las acciones oportunas. A esa variable la llamaremos error. El aspecto final de la clase Dios sería algo así:

```
lass Dios
{

public :

Dios (const String& n) : elNombre (n), elError (0)
{ if (Dios::dios (elNombre) != NULL)
{ elError = 1; return; }
  else Dios::losDioses.insertar (this); }
~Dios ();
const String& nombre (void) const;
virtual ResultadoAccion actuar (void) = 0;
int error (void) const { return (elError); }
static Dios* unDios (void);
static Dios* dios (const String&);

protected :

String elNombre;
int elError;
static Lista<Dios*> losDioses;

};

la cual puede considerarse un buen diseño.
c
```

DISEÑANDO JUEGO CONVERSACIONAL

Juego conversacional es la clase, como ya se ha dicho anteriormente, que controla toda la ejecución del juego. Parece claro que, al menos, debe tener tres métodos fundamentales: inicializar, jugar y finalizar. El primero se debe encargar de cosas como crear el escenario del juego (el mundo, los dioses, etc...), el segundo es el eje central del sistema, mientras que el tercero y último decide cosas como cuando finalizar el juego y la manera de hacerlo. Ni que decir tiene que son métodos virtuales ya que cada juego tendrá una forma distinta de ejecutarse.

Dentro de esta clase figurarán, además, todas aquellas cosas que sean de interés general dentro del juego: el mundo imaginario, los jugadores principales, etc...

NUEVAMENTE LA UNICIDAD

La clase Juego Conversacional es, nuevamente, una clase única. Sin embargo, tiene una complicación adicional. Al ser una clase en la que se almacena tanta información necesaria para el normal funcionamiento de

```
class JuegoConversacional
{

public :

virtual ResultadoAccion inicializar (void) = 0;
virtual ResultadoAccion jugar (void) = 0;
virtual ResultadoAccion finalizar (void) = 0;
MundoImaginario* mundo (void) { return
(elMundo); }
...

protected :

MundoImaginario *elMundo;

};
```

las aplicaciones conversacionales tendrá que ser accedida desde muy diversos puntos (todos los dioses, por ejemplo, necesitarán conocer cual es el mundo sobre el que deben actuar). No parece una solución muy orientada o objetos el construir la instancia de Juego Conversacional mediante una variable global accesible por cualquier módulo (aunque es una solución muy rápida). ¿Qué hacer? pues lo mismo que antes.

```
class JuegoConversacional
{

public :

JuegoConversacional () : elError (0)
{ if (elJuego != NULL)
{ elError = 1; return; }
  else elJuego = this;
}
virtual JuegoConversacional () { };
virtual ResultadoAccion inicializar (void) = 0;
virtual ResultadoAccion jugar (void) = 0;
virtual ResultadoAccion finalizar (void) = 0;
MundoImaginario* mundo (void) { return
(elMundo); }
int error (void) const { return (elError; }
static JuegoConversacional* juego (void)
{ return (elJuego); }
...

protected :

MundoImaginario *elMundo;
int elError;
static JuegoConversacional* elJuego;

};
```

Dentro de la clase colocamos una variable estática que apunte al único objeto de esta clase que exista y un método, también estático, para acceder a ella. Cuando el primero objeto de la clase sea creado, esta variable le señala y cualquier otro intento de construcción es rechazado por la misma vía que antes:

de esta forma se garantiza un objeto único fácilmente accesible desde cualquier punto de la aplicación.

ALGUNOS HILOS

La serie de artículos sobre juegos conversacionales ha tenido, desde el principio, tres objetivos fundamentales: el primero de ellos asentar los conocimientos de C++ obtenidos durante otras etapas anteriores y; el segundo, hacerle ver que la programación orientada a objetos es algo más complejo que la clásica funcional y que se pierde mucho más tiempo analizando los problemas que escribiendo el código para resolverlos y, el tercero y último, indicarle el camino para construir una librería de clases para el desarrollo de juegos conversacionales.

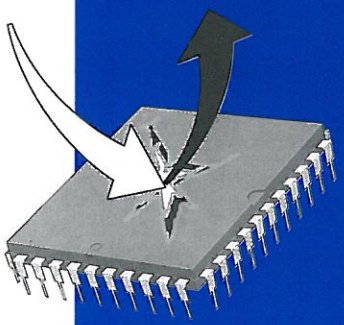
La exposición, por tanto, no pretendía ser exhaustiva respecto de las clases que deberían componer esa librería. Así, han aparecido clases como: *ResultadoAccion* o *Lista* ni siquiera enunciadas. En todas estas omisiones se ha considerado que el nombre era más que suficiente para señalar la funcionalidad cumplida y que entrar en más detalle retrasaría inútilmente el resto de los objetivos. Esperamos que las decisiones hayan sido tomadas con acierto.

CONCLUSIÓN FINAL

Como conclusión fundamental indicarle que, piense siempre los problemas antes de pasar a su resolución, los escriba sobre un papel y vea la forma de implementarlos en una máquina (persistencia). Durante esta última fase, muchos de los problemas se repetirán y, si eligió un criterio adecuado de resolución, podrá reusarlo muy eficaz y rápidamente (como ha podido observar con el caso de la unicidad mostrado en este ejemplo). No se preocupe si el tiempo de análisis y diseño de un problema se alarga, es normal en orientación a objetos. Si lo hace bien, el de codificación bajará en picado.

Aprovecho la ocasión para, personalmente y en nombre de la revista, agradecerle la atención prestado. Espero que nos podamos volver a encontrar nuevamente. Un afectuoso saludo.

ANALIZADORES SINTÁCTICOS (yII)



Daniel Navarro

Se continúa en este número con la implementación de un analizador sintáctico para el lenguaje Letra, viéndose como en este punto es donde se toman las decisiones más importantes referentes al compilador de Letra. Para comenzar se hará un breve repaso de lo visto hasta este punto en esta serie y de la situación en la que se detuvo el compilador el mes pasado.

Se explicó la estructura y funcionamiento general de un compilador en el primer artículo. En el siguiente se definió el lenguaje a nivel de usuario, implementado su analizador léxico en el tercer artículo, que era el encargado de leer el fichero de entrada (un programa cualquiera en Letra) e irnos informando de los elementos o "piezas" que se iba encontrando. Y en el último artículo se definió su sintaxis con la gramática BNF (sintaxis que aparece, ahora ampliada, en el listado 1) y se explicó como, a partir de esta, se podía crear un analizador sintáctico para Letra, encargado de comprobar que un programa se adaptaba a esta sintaxis definida.

La sintaxis se expresaba utilizando unas reglas que iban definiendo diferentes partes del programa; a estas partes se les daba un nombre y se decía que eran símbolos no terminales. Podíamos definir por ejemplo, reglas sintácticas como la siguiente:

<SENT_MIENTRAS> ::= p_mientras <EXPRESION> { <SENTENCIA> } p_fin

Queriendo decir que la parte del programa denominada *<SENT_MIENTRAS>* (una sentencia similar a while (...)) se definía como la pieza p_mientras (que hace referencia en nuestra lista de piezas a la palabra reservada MIENTRAS) seguida de otra parte del programa denominada <EXPRESION> (que debe ser definida con otra regla sintáctica), pudiendo entonces aparecer a continuación desde ninguna a muchas <SENTENCIA> (que es la parte de programa que hace referencia a una sentencia cualquiera) y por último debía venir la pieza p_fin (palabra reservada FIN). Por ejemplo, una *<SENT_MIENTRAS>* en un programa podría ser algo como lo siguiente:

```
mientras (a<100)
  a=a+1
  b=(b+a)/2
fin
```

Para implementar posteriormente un analizador para el lenguaje, se definían procedimientos de análisis que se correspondieran con las diferentes reglas sintácticas, es decir, se construiría un procedimiento llamado *an_SENT_MIENTRAS()* para analizar las sentencias mientras, un procedimiento llamado *an_EXPRESIÓN()* para analizar expresiones, etc.

El analizador léxico era el que leía el fuente y nos iba diciendo, indicándolo en una variable denominada pieza_actual, cuales eran las piezas que se encontraba en el fichero. A los códigos de las diferentes piezas que el analizador léxico podía reconocer se les había dado un nombre que empezaba siempre por p_ (estos aparecen en el fichero "lexico.h").

El éxito o fracaso de un lenguaje de programación tiene mucho que ver con la calidad de sus primeras implementaciones, un mal compilador para un buen lenguaje puede hundir su reputación definitivamente.

Y el trabajo del analizador sintáctico era comprobar para cada pieza que venía si era correcta, o mejor dicho, si podía aparecer una pieza de ese tipo en la parte del programa que se estaba analizando en ese momento. Si la pieza estaba bien emplazada, el analizador sintáctico llamaría entonces al analizador léxico para que este leyera la siguiente pieza del programa (usando la función Obtener_pieza()), y en caso contrario se emitiría un error abortando el proceso de compilación.

ACUERDOS EN LA IMPLEMENTACIÓN

Los diferentes procedimientos de análisis sintáctico se pueden implementar de muchas formas, pero aquí se tomará el siguiente convenio:

Cuando un procedimiento se disponga a analizar cualquier parte del programa, este será invocado con la primera pieza ya leída (en pieza_actual). Una vez que el procedimiento determine que una pieza es correcta y esta ya no se necesite más, se leerá la siguiente, por lo que todos los procedimientos de análisis terminarán con la siguiente pieza leída, incluso, cuando no necesiten dicha pieza para nada.

Para comprobar que a continuación debe venir una pieza determinada, por ejemplo de tipo p_entero, se utilizará una sentencia como la siguiente:

```
if (pieza_actual != p_entero) error();
// Aquí se tratará en un futuro el número entero
obtener_pieza();
```

Por tanto un procedimiento como el que implementa la regla que definía <SENT_MIENTRAS>, quedará de la siguiente forma:

```
// Regla BNF de <SENT_MIENTRAS> ::=
// p_mientras <EXPRESION> { <SENTENCIA> } p_fin

void an_SENT_MIENTRAS(void)
{
   if (pieza_actual != p_mientras) error();
  obtener_pieza();
  an_EXPRESION();
  while (pieza != p_fin) an_SENTENCIA();
  obtener_pieza();
}
```

Este procedimiento analizará todas las sentencias mientras que aparezcan en los programas, pero ¿Cuándo será llamado?, pues bien, si miramos el listado 1 podremos observar que la primera regla define lo que es un <PROGRAMA>, y por consiguiente el primer procedimiento de análisis invocado será *an_PROGRAMA()*. En algún momento este procedimiento determinará que va a venir una <SENTENCIA> llamando entonces a *an_SENTENCIA()*, que es un procedimiento de análisis de una sentencia cualquiera, pero que se apoya en otros muchos procedimientos (pues su regla BNF se apoya en otras posteriores) que son los encargados de analizar las diferentes sentencias en concreto. De modo que cuando suponga que va a venir una sentencia mientras, porque venga una pieza p_mientras, llamará al procedimiento *an_SENT_MIENTRAS()*.

Los procedimientos de análisis se corresponden directamente con las reglas sintácticas

Es muy normal encontrar casos como el anterior; cuando se llama a *an_SENT_MIENTRAS()* es porque ha venido una pieza de tipo p_mientras, sin embargo, lo primero que este comprueba (pues su cometido es asegurar que viene una sentencia mientras completa y bien construida) es que la primera pieza sea p_mientras y en caso contrario emitir un error. Obviamente no emitirá nunca dicho error, por lo que esta primera comprobación será omitida. Sin embargo se debe tener mucho cuidado, pues hay ocasiones en las que se puede llamar a un procedimiento de análisis desde varios puntos, sabiendo en algunos que su primera pieza es correcta pero desconociéndolo en otros; en dicho caso el procedimiento no puede omitir comprobar su primera pieza.

PEQUEÑOS ARREGLOS

Se han introducido pequeñas modificaciones en la BNF de Letra con respecto al mes pasado, que no alteran en absoluto el lenguaje, pero simplifican la estructuración del compilador.

Otra modificación ha sido añadir en la sintaxis del lenguaje las funciones especiales. Esto no se quiso hacer antes porque realmente estas funciones no forman parte del lenguaje, ya que podrían ser cambiadas por otras cualquiera. Aun así se han tenido que añadir para que el compilador pueda tratar, desde ya, los programas en Letra que utilizaban dichas funciones.

Igualmente se ha subsanado un pequeño desliz: el analizador léxico cuando era inicializado mediante la función inicializacion_lexica() reservaba memoria para cargar el fichero y esta no se liberaba en ninguna parte. Ahora esto lo hace un nuevo procedimiento léxico, llamado finalizacion_lexica(). Hay otros detalles, como suponer que el fichero puede ser cargado entero en memoria de una sola vez, que no son errores, sino simplificaciones para facilitar la comprensión del código del compilador.

Se puede observar como queda este analizador completo en el fichero *AN_SINTA.CPP* que se incluye en el disco de la revista. Este lo podemos utilizar para analizar cualquiera de los programas que teníamos como ejemplos de Letra y nos informará sobre si estos son o no correctos, tanto léxica como sintácticamente.

MENSAJES DE ERROR

La función que emite un error aborta el proceso de compilación con un *exit()*, pero antes debe imprimir suficiente información en pantalla como para que el usuario pueda corregir el error detectado, por lo que no bastará con *"Hay un error en la línea 333"*.

La gran mayoría de los errores sintácticos se producen porque no ha venido la pieza que se esperaba, por lo que se ha implementado una función genérica de error a la que, cuando se produce un error, se le pasa la pieza que debería haber venido, y la función emite un error como el siguiente:

Error sintáctico en la línea N, se esperaba A en lugar de B.

Sin embargo a veces no se espera una pieza en concreto, sino que hay varias posibilidades, como cuando se espera un factor en una expresión o que empiece una sentencia cualquiera. A estos errores se les ha dado un código numérico, para que cuando la función de error lo reciba como parámetro, informe de dicho error.

Cómo se detectan los errores sintácticos tiene mucho que ver con cómo se construye la sintaxis con la gramática BNF, y cómo después es interpretada. Por ejemplo, si pretendemos analizar el símbolo no terminal <S> definido de esta forma:

```
<S> ::= <R> | p_a
<R> ::= p_b
```

El procedimiento an_S se podría construir de cualquiera de las dos siguientes formas, desembocando cada una de ellas en un error diferente cuando al analizar <S> nos encontráramos con una pieza no esperada como pudiera ser p_x.

```
void an_S(void) {
  if (pieza_actual == p_b) an_R();
  else {
    if (pieza_actual != p_a) error();
    obtener_pieza();
  }
}
```

```
void an_S(void) {
  if (pieza_actual != p_a) an_R();
  else {
    obtener_pieza();
  }
}
```

En el primer caso se emitirá un error *"Se esperaba una pieza p_a en lugar de p_x"* y en el segundo, el error que se detectará dentro de an_R() será del tipo *"Se esperaba una pieza p_b en lugar de p_x"*. No obstante, una buena implementación del lenguaje debería emitir un error *"Ha llegado p_x y se esperaba que llegara p_a o p_b"*.

CONTINUANDO EL COMPILADOR.

Por ahora dejaremos el compilador detenido, completando el analizador sintáctico para Letra. Como ya se ha mencionado anteriormente, este analizador es la estructura ósea de todo el compilador, esto quiere decir que cuando continuemos con el compilador con el análisis semántico o la generación de código, lo realizaremos añadiendo "cosas" en el analizador sintáctico. Para finalizar la fase de análisis sólo nos queda el análisis semántico.

Los errores sintácticos se producen cuando llega una pieza no esperada

¿Qué cosas quedan por comprobar para saber que un programa es correcto?, si nos detenemos un momento, nos daremos cuenta que todavía no se están comprobando muchas cosas:

- si se inicializa una tabla con más valores de los que cogen en ella
- si se define un dato dos o más veces
- si utilizamos algún dato o una etiqueta que no se ha definido en ningún sitio
- si definimos un dato y luego lo utilizamos como si fuera una tabla
- si existe memoria suficiente para todos los datos y tablas que hemos definido
- etc.

Este será el trabajo del analizador semántico. Los únicos puntos entonces, que faltarán por comprobar en los programas, serán los errores en tiempo de ejecución, como divisiones por cero o desbordamientos de pila. Dichos errores no se pueden comprobar en el análisis, pero el generador de código introducirá en los programas, mecanismos para que dichos errores puedan ser autodetectados en tiempo de ejecución.

INVENTANDO OTROS LENGUAJES.

Quizá este sea el mejor momento para que el lector se proponga hacer alguna variación en el diseño de Letra e intente llevar a buen fin el compilador para su exclusivo lenguaje, a la vez que aquí se va terminando el compilador de Letra. Para ello debe tener en cuenta que cuanto más se aparte del tipo de lenguajes a los que representa Letra, más difícil le será conseguirlo.

Lo primero que se debe definir es la finalidad del lenguaje que vamos a crear, después definir, como se hizo aquí, el lenguaje a nivel de usuario, o mejor dicho imaginárselo. Tras ello crear una lista con todas las piezas del lenguaje e implementar un analizador léxico que se encargue de reconocerlas. Entonces, en base al diseño del lenguaje, se deberá especificar su sintaxis, bien sea con BNF o con grafos. Implementar el analizador sintáctico no será muy difícil si se observa el ejemplo que aquí se presenta, y una vez hecho esto ya se tendrá el compilador a medio construir, por lo que puede intentar avanzar antes de que lo hagamos con Letra.

Es importante que se defina nuestro lenguaje de forma que cada vez que puedan ser invocados diferentes procedimientos, como en el caso de <SENTENCIA>, todos ellos deban empezar por piezas diferentes (al igual que en Letra) para así poder averiguar a que procedimiento de análisis se llamará en función de cual sea la siguiente pieza que aparece en el programa. Si el lenguaje no fuera diseñado de esa forma también se podría, por supuesto, construir un compilador para él, pero resultaría algo más complejo porque se debería tener siempre información sobre cuales son las "n" siguientes piezas que vienen en el programa para poder determinar las ramificaciones que va tomando el analizador. Esto se conseguiría separando el análisis léxico y el sintáctico, es decir, primero ejecu-

tando el analizador léxico para obtener una lista con todas las piezas del programa, y cuando estuviera completa dicha lista comenzaría el trabajo del analizador sintáctico, con lo que este podría resolver situaciones como:

<S> ::= <Q> | <R>
<Q> ::= {p_a} p_b
<R> ::= {p_a} p_z

Ya que cuando estuviera analizando <S> podría determinar si va a llamar a <Q> o a <R>, para ello miraría tras todas las piezas p_a que vinieran, que viene a continuación, si una pieza p_b o una pieza p_z y en función de ello llamaría a an_Q() o bien a an_R().

En el otro extremo, quienes se sientan más intimidados por la complejidad que puede suponer un compilador, pueden intentar seguir el desarrollo de Letra (al cual tampoco le queda ya mucho) cambiando simplemente las funciones especiales de este por otras suyas propias. Se pueden crear variantes del lenguaje con posibilidades gráficas, sonoras o de cualquier otro tipo.

INTRODUCCIÓN AL ANÁLISIS SEMÁNTICO.

En la última fase de análisis el trabajo a realizar será el mismo que en las anteriores, comprobar que un programa sea correcto, pero aquí todas las comprobaciones estarán relacionadas con los objetos que se puedan definir en el lenguaje.

Todos los lenguajes de programación permiten que el programador se invente objetos de un tipo determinado, como pueden ser variables o procedimientos, y les asigne a su antojo un nombre (identificador) y unas características. El compilador debe tomar buena nota de todos los objetos, y esto lo hará en la llamada "Tabla de objetos".

Dicha tabla se divide en un número campos en los que se especificarán todas las características del objeto. Por poner un ejemplo, cuando el compilador de Letra se encuentre una declaración de una tabla como:

mi_tabla[10]=0:1:2:3:4:5:6:7:8:9

Podría guardar en la tabla de objetos un nuevo registro con la siguiente información:

- Tipo de objeto: Tabla
- Nombre: "mi_tabla"
- Posición de memoria asignada: x
- Número de posiciones: 10
- Está declarado: Sí
- Ha sido usado: No
- Etc.

Normalmente el primer procedimiento de análisis invocado será *an_PROGRAMA()*

Parte de esta información no corresponderá al analizador semántico, sino a otras partes del compilador como el generador de código, por ejemplo, al analizador no le interesa la posición de memoria en la que se guardará la tabla, sólo es útil para generar el código de las instrucciones que accedan a ella.

El analizador semántico accederá a la tabla de objetos, tanto para guardar información (al declarar una variable), como para consultarla (cuando se utilice en el resto del programa dicha variable). Y se emitirán los errores del siguiente nivel, los errores semánticos.

PRÓXIMO NÚMERO

En Abril se estudiarán los problemas que acarrea un analizador semántico, y entonces se implementará uno para Letra. Tras esto se habrá finalizado todo el análisis, quedando sólo por ver la última fase, la síntesis, o dicho de otra forma: la generación de código.

PREGUNTAS Y RESPUESTAS

¿Por qué no se ha hecho un compilador para C?

El leguaje y el compilador que aquí se plantean son mucho más sencillos que los comerciales actualmente empleados, un compilador para C es bastante más complejo, sobre todo en cuanto a la calidad del código generado. También existen muchos lenguajes más sencillos que Letra usados comercialmente, son lenguajes que pertenecen a muchas aplicaciones, y que se usan para crear ficheros "script", de configuración de algún sistema, o bien lenguajes de macros. Por razones educativas, aquí se ha optado por algo intermedio.

¿El compilador que se está haciendo, es el mismo que se entregó al principio?

No, el compilador que se entregó como ejecutable al inicio de la serie (y que, para quienes no lo tengan, se vuelve a entregar ahora en el fichero C_LETRA.ZIP) era otra implementación de este lenguaje diferente a la que aquí se está desarrollando, pues estaba realizada en PASCAL y de una forma menos estructurada. Aún así, ambas versiones funcionarán de igual forma una vez completadas.

¿Se explicará también cómo hacer el entorno de programación?

No, el objetivo de esta serie es simplemente completar un compilador externo (en línea de comandos) para Letra que produzca ficheros ejecutables (.COM). El como crear un editor de programas es algo que no se piensa abordar, al menos en un principio.

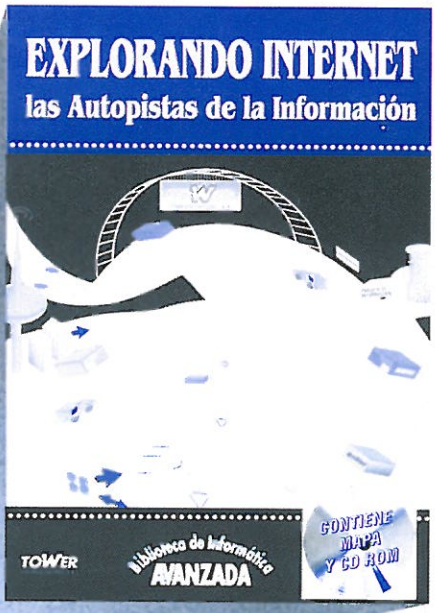
¿Dónde se puede ampliar más información para hacer un compilador?

Se está preparando una bibliografía. En las carreras de informática, tanto técnicas como superiores, se aborda el tema de los compiladores como una asignatura, de una forma mucho más teórica y completa, aunque menos práctica, pues su objetivo es sentar las bases para el desarrollo de un compilador real. Aun así, la información y bibliografía utilizada en dichas carreras, es perfectamente válida para ampliar los temas aqui expuestos.

Cuadro 1.

LOS LIBROS DE INFORMÁTICA MÁS ACTUALES AL MEJOR PRECIO

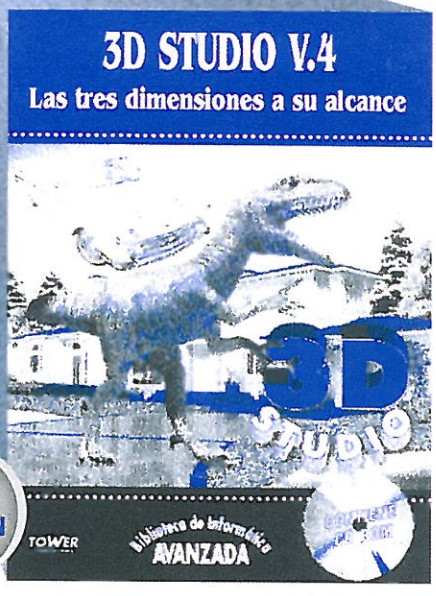
Contiene CD-ROM con una selección de ficheros de sonido, vídeo e imágenes, así como las utilidades disponibles actualmente para Windows 95.



Contiene CD-ROM con 800 programas de acceso rápido a INTERNET

CADA LIBRO POR SÓLO:

1.995 ptas.
IVA INCLUIDO



TOWER COMMUNICATIONS

A LA VENTA EN QUIOSCOS, TIENDAS ESPECIALIZADAS O LLAMANDO AL (91) 661 42 11

SINTAXIS AMPLIADA DE LETRA

```
# Sintaxis BNF de Letra ampliada y comentada, esta sintaxis no es ningún pro-
# grama (aunque parezca uno) es solo una forma de estructurar nuestro lenguaje
# de programación para poder crear un compilador para él fácilmente.
# Se define un programa como una sucesión de cualquier número de sentencias
# (incluso cero) y a continuación de ellas la pieza p_eof (fin de fichero)
<PROGRAMA> ::= { <SENTENCIA> } p_eof
# Se define una sentencia como uno cualquiera de los siguientes elementos.
<SENTENCIA> ::=
  <ETIQUETA> | <DECLARACION_DATOS> | <ASIGNACION> | <SENT_SI> |
  <SENT_MIENTRAS> | <SENT_REPETIR> | <SENT_IR> | <SENT_HACER> |
  <SENT_VOLVER> | <FUNCION_INTERNA>
# Una etiqueta no será más que una pieza de tipo etiqueta (estas son reconocidas
# por el analizador léxico, pues son nombres que empiezan con un punto).
<ETIQUETA> ::= p_etiqueta
# Una sentencia "declaración de datos" se construye comenzando con la palabra
# reservada DATOS y finaliza con FIN; entre estas dos puede aparecer cualquier
# número de declaraciones de datos, que empiezan todas con un identificador
# (el nombre de la variable) y a continuación, de manera opcional, puede apare-
# cer una declaración de este dato para inicializarlo o bien para indicar que
# se trata de una tabla de datos.
<DECLARACION_DATOS> ::=
  p_datos
  { p_identificador [ <INICIALIZACION_DATO> | <INICIALIZACION_TABLA> ] }
  p_fin
# Una sentencia de asignación se define como una referencia a un dato (que
# podrá ser una simple variable o bien una tabla con su índice) seguida siempre
# del signo igual (=) y una expresión cualquiera.
<ASIGNACION> ::= <DATO> p_igual <EXPRESION>
# La sentencia si (más conocida como if ... else ... endif) será la palabra
# reservada SI y una expresión, tras las cuales puede aparecer cualquier número
# de sentencias y entonces la palabra FIN (finalizando la sentencia) o bien la
# palabra SINO, otra serie de sentencias y la palabra FIN.
<SENT_SI> ::= p_si <EXPRESION> { <SENTENCIA> } [ p_sino { <SENTENCIA> } ] p_fin
# La sentencia mientras (while), compuesta de la palabra reservada MIENTRAS, una
# expresión, cualquier número de sentencias y FIN.
<SENT_MIENTRAS> ::= p_mientras <EXPRESION> { <SENTENCIA> } p_fin
# Sentencia repetir (do ... while, o bien repeat ... until), REPETIR seguido
# de una serie de sentencias y entonces HASTA seguido de una expresión.
<SENT_REPETIR> ::= p_repetir { <SENTENCIA> } p_hasta <EXPRESION>
# Sentencia ir (goto), la palabra reservada IR seguida de una etiqueta
<SENT_IR> ::= p_ir p_etiqueta
# Sentencia hacer (gosub), HACER seguido de una etiqueta
<SENT_HACER> ::= p_hacer p_etiqueta
# Sentencia volver (return), simplemente la palabra reservada VOLVER
<SENT_VOLVER> ::= p_volver
# La inicialización de un dato definido en la zona de datos se podrá hacer
# poniendo un signo igual y después una constante.
<INICIALIZACION_DATO> ::= p_igual <CONSTANTE>
# Para inicializar una tabla de datos deberemos indicar entre corchetes su
# longitud (con un entero) pudiendo después, opcionalmente, indicar el conte-
# nido de la tabla con un signo igual y varias constantes separadas por sím-
# bolos dos puntos (:). Otra posible definición admitida consiste en definir una tabla
# omitiendo su longitud entre los corchetes, pero en dicho caso será obligato-
# ria la inicialización de la tabla (para poder calcular su longitud).
<INICIALIZACION_TABLA> ::=
  p_abrir_corchete
    (p_entero p_cerrar_corchete
    [ p_igual <CONSTANTE> { p_dos_puntos <CONSTANTE> } ] |
    p_cerrar_corchete p_igual <CONSTANTE> { p_dos_puntos <CONSTANTE> } )
# Una constante puede ser, las palabras reservadas CIERTO o FALSO (true, false)
# un entero (opcionalmente precedido de un + o un -) o bien un literal (un
# texto entre comillas).
<CONSTANTE> ::= p_cierto | p_falso | [ p_mas | p_menos ] p_entero | p_literal
# Una referencia a un dato es simplemente el nombre de ese dato, y si es una
# tabla debemos indicar entre corchetes a que posición de ella nos referimos
# utilizando una expresión
<DATO> ::= p_identificador [ p_abrir_corchete <EXPRESION> p_cerrar_corchete ]
# Una expresión se define como un cómputo (una expresión sin operadores
# lógicos), tras la cual, mientras venga un operador lógico, pueden seguirle
# mas cómputos.
<EXPRESION> ::= <COMPUTO> { <OP_LOGICO> <COMPUTO> }
# Un computo se define igualmente como un valor (expresión sin operadores
# lógicos ni de comparación) o varios separados por operadores de comparación.
<COMPUTO> ::= <VALOR> { <OP_COMPARACION> <VALOR> }
# Un valor será un término (expresión que solo puede utilizar los operadores
# binarios de mayor prioridad: multiplicar, dividir o módulo) o varios,
# separados por operadores del nivel aditivo (suma y resta)
<VALOR> ::= <TERMINO> { <OP_ADITIVO> <TERMINO> }
# Un término será un factor (el elemento básico de una expresión) o varios
# separados por operadores del nivel multiplicativo).
<TERMINO> ::= <FACTOR> { <OP_MULTIPLICATIVO> <FACTOR> }
# Un factor, es el elemento básico de una expresión como son: CIERTO, FALSO, un
# número entero, un literal, una referencia a un dato (o tabla) o bien otra
# expresión completa entre paréntesis. A estos elementos les puede preceder un
# operador unario (o varios), como los signos positivo y negativo o el operador
# NO de negación (el not lógico).
<FACTOR> ::=
  <OP_UNARIO> <FACTOR> | p_abrir_parentesis <EXPRESION> p_cerrar_parentesis |
  <DATO> | p_cierto | p_falso | p_entero | p_literal
# Operadores de nivel lógico Y (and) y O (or), son los operadores de menor
# prioridad en el cálculo.
<OP_LOGICO> ::= p_y | p_o
# A estos les siguen los operadores de comparación siguientes.
<OP_COMPARACION> ::=
  p_mayor | p_mayor_igual | p_menor | p_menor_igual | p_igual | p_distinto
# Los operadores de nivel aditivo son solo la suma y la resta.
<OP_ADITIVO> ::= p_mas | p_menos
# Los de nivel multiplicativo, el signo de multiplicar, de dividir y el de
# módulo (o resto de una división entera) son los operadores binarios de mayor
# prioridad (binario quiere decir que opera con los elementos que tiene a su
# izquierda y a su derecha, obteniendo un resultado de ambos).

<OP_MULTIPLICATIVO> ::= p_multiplicar | p_dividir | p_modulo

# Los operadores unarios afectan directamente (y únicamente) al factor que
# tienen a continuación, por ello son de mayor prioridad que los binarios.

<OP_UNARIO> ::= p_mas | p_menos | p_no

# Por último definiremos las funciones internas del lenguaje, que actúan como
# si se tratara de otras sentencias, y son las siguientes.

<FUNCION_INTERNA> ::=
  <BORRAR_PANTALLA> | <ESCRIBIR> | <LEER_TECLA> | <MOVER_CURSOR> |
  <COLOR> | <ALEATORIO> | <ESPERAR> | <LEER_CARACTER> | <PONER_CARAC-
TER>

# Todas las funciones se definen como una palabra reservada determinada, que
# es el nombre de la función, y después entre paréntesis los parámetros que
# se requieran. Las funciones que requieren como parámetro un <DATO> (referen-
# cia a un dato o posición de tabla) es por que van a devolver un valor en
# dicho dato. Estas sentencias tienen la siguiente sintaxis:

<BORRAR_PANTALLA> ::= p_borrar_pantalla p_abrir_parentesis p_cerrar_parentesis

<ESCRIBIR> ::= p_escribir p_abrir_parentesis p_literal p_cerrar_parentesis

<LEER_TECLA> ::= p_leer_tecla p_abrir_parentesis <DATO> p_cerrar_parentesis

<MOVER_CURSOR> ::=
  p_mover_cursor p_abrir_parentesis <EXPRESION> p_coma
  <EXPRESION> p_cerrar_parentesis

<COLOR> ::= p_color p_abrir_parentesis <EXPRESION> p_cerrar_parentesis

<ALEATORIO> ::= p_aleatorio p_abrir_parentesis <DATO> p_cerrar_parentesis

<ESPERAR> ::= p_esperar p_abrir_parentesis <EXPRESION> p_cerrar_parentesis

<LEER_CARACTER> ::=
  p_leer_caracter p_abrir_parentesis <DATO> p_cerrar_parentesis

<PONER_CARACTER> ::=
  p_poner_caracter p_abrir_parentesis <EXPRESION> p_cerrar_parentesis
```

Listado 1.

RAY TRACING (VI): FICHEROS DE CONFIGURACIÓN

Álvaro Silgado

En este capítulo se ofrece una nueva herramienta que va a ser muy útil : la posibilidad de leer la configuración de la escena que se quiere obtener desde un fichero ascii.

A lo largo de este curso, se ha ido siguiendo paso a paso todos los aspectos de un programa de Ray Tracing. Se ha podido ir obteniendo imágenes verdaderamente complejas y con acabados muy realistas según ha aumentando la complejidad del programa. Pero éste no era cómodo de usar : cada escena que se pretendía confeccionar suponía una modificación del código, puesto que no había otra forma de dar de alta los objetos en la escena.

En el presente capítulo se abordará la posibilidad de escribir la configuración de la escena en un fichero ascii que, posteriormente, será leído por el programa generando de esta forma su imagen asociada.

Cabe advertir de entrada que dicha utilidad se aleja un poco de la programación gráfica, acercándose un poco más a la programación de compiladores, ya que de lo que se trata es de leer un "código fuente", "compilarlo" (entender qué es lo que pone en él) y "ejecutarlo" (generar la imagen tal y como está descrita en el fichero".

Una última advertencia : hay algunos aspectos del compilador que han tenido que ser sacrificados dada su complejidad a la hora de leer un fichero. El ejemplo más evidente son los patrones. Permitir leer una función de decisión supone construir un compilador de C que entienda en qué casos tiene que aplicar un material u otro. Dicho sacrificio será compensado por un aumento de la versatilidad del programa y de su facilidad de uso.

ENSEÑAR A LEER

Enseñar a leer a un programa no es tarea fácil, ni mucho menos. Lo primero que hay que decidir es qué tipo de lenguaje se va a enseñar a leer. Para ello, se ha diseñado un lenguaje de "programación de gráficos", cuya estructura se aprecia en la figura 1.

La normas básicas del fichero de configuración son las siguientes :

- Se puede poner comentarios en cualquier punto del fichero, siempre que no partan una palabra o número por la mitad. Los comentarios comienzan con los caracteres /* y terminan con los caracteres */, como en lenguaje C.
- Para definir un objeto, se pone su nombre y a continuación su contenido. Si el objeto está formado por un único campo (por ejemplo, el campo "espejo"), simplemente se escribe su valor a continuación. Si está formado por varios campos, se abre una llave "{" a continuación del nombre de objeto, se definen uno por uno sus campos (que también pueden estar formados por varios campos) y se cierra la definición con otra llave, pero de cerrar ("}").
- Siempre debe existir una cámara correctamente definida y se debe especificar un nombre de fichero.
- Los campos de los objetos pueden aparecer o no. Si no se especifican, se considera su valor por defecto (generalmente, 0).
- Se pueden definir tantos objetos como se quiera (siempre que la memoria del ordenador lo permita).

LEER CON UN LECTOR

Una vez definido lo que se quiere leer, hay que implementar el proceso de lectura. Para ello, se ha creado un objeto "lector" que simplifica la tarea de escrutar el fichero ascii, separando las palabras de los números, ignorando los comentarios y saltándose los espacios en blanco y retornos de línea.

El método constructor del lector recibe como parámetro de entrada el nombre del fichero que se quiere leer. Entonces, intenta abrirlo y lee de él sus primeros bytes, guardándolos en un buffer propio. Si se produce un error en cualquiera de estos procesos, se aborta la ejecución del programa. Un segundo método sirve para cerrar el fichero abierto una vez finalizada su lectura.

El tercer método, leer_buffer, sirve para leer la siguiente ristra de bytes del fichero, convirtiendo todos sus caracteres en mayúsculas, guardándola en el buffer y actualizando los punteros asociados. Actualiza una variable llamada *max_indice*, que indica el último carácter leído en el buffer. Si esta función devuelve dicha variable con valor 0, es que se ha alcanzado el final de fichero. El cuarto, es un simple mensaje de error de sintaxis que implica la terminación de la ejecución del programa de manera inmediata.

El siguiente método sirve para saltarse los espacios y los comentarios, y se va a estudiar más detalladamente.

IGNORAR EL ESPACIO ... DE MÁS

Para poder saltar los espacios y los comentarios es necesario conocer un poco más a fondo la estructura del lector. Como se ha dicho antes, está formado por un buffer interno en el que se van haciendo lecturas. Esto se hace así, porque leer carácter a carácter de un fichero es muy lento y no permite retroceder, en caso de que se necesitara. Para saber a qué carácter acceder, hay un puntero llamado *indice* que hace referencia al carácter que toca leer. Además, hay dos variables, llamadas *fila* y *columna*, que sirven para que, en caso de existir un error sintáctico, indicar en dónde se ha producido.

Así, cada vez que se lee un carácter, se incrementa la variable columna. Si ese carácter es un retorno de carro, se pone a 1, y si es un avance de carro, se incrementa la variable fila. En el caso de ser un tabulador, se calcula la siguiente columna múltiplo de 8 y ese es el valor que se asigna a la variable. De esta forma, saltarse los espacios supone ir incrementando uno a uno el índice del buffer, y continuar hasta encontrar un carácter que no sea ni blanco, ni retorno de carro ni avance del carro.

FIGURA 1

```
/* Comentario : (puede estar en cualquier sitio y haber tantos como se quiera).

Este es un ejemplo de fichero de configuración. Todos los campos pueden aparecer o no, según se desee (aunque algunos, como la cámara o el nombre del fichero, son obligatorios) y en cualquier orden. */

Camara /* Los texto pueden ir en mayúsculas o en minúsculas. */
{ /* Las llaves pueden aparecer en cualquier sitio. */
Desde { 120, 50, 30 } /* Posición de la cámara. "x", "y" y "z" son sus coordenadas. */
Hacia { -5, 20, -10 } /* Punto hacia el cual mira la cámara. */
Pixels_x 640 /* Tamaño horizontal de la ventana a obtener. */
Pixels_y 480 /* Tamaño vertical de la ventana a obtener. */
Angulo_f 45 /* Ángulo focal de la cámara, en grados decimales. */
}

Ambiental { 60, 60, 60 } /* Definición de la luz ambiental. */

Fichero bolas.tga /* Nombre del fichero a generar. */

Cielo /* Definición del objeto cielo. */
{
Material /* Definición del material del que está compuesto. */
{
Difusa { 0.2, 0.2, 0.6 } /* Luz difusa. No tiene sentido usarla si se emplea una textura. */
Kr 0.2 /* Coeficiente de refracción. No se usa en el cielo. */
Ks 0.3 /* Coeficiente de luz especular. No se usa en el cielo. */
Ps 20 /* Potencial especula. Tampoco se emplea en el cielo. */
Kt 0.5 /* Coeficiente de luz transmitida. No se usa en el cielo. */
Color nubes.pcx /* Textura de color. Si no se especifica, no se usa. */
Intersec franjas.pcx /* Textura de intersección. Si no se especifica, no se usa. */
Escala_x 1 /* Escalado horizontal de la textura. */
Escala_y 1 /* Escalado vertical de la textura. */
Espejo 1 /* Espejo de textura o no. */
            }
}

Suelo /* Definición de la clase suelo. */
{
Luminoso 0 /* Especifica si es luminoso o no. */
Material
{
..... /* Aquí iría una definición similar a la anterior. */
            }
}

Poligono /* Definición de un objeto polígono. */
{
Punto { -80, 60, 10 } /* Primer punto del polígono. */
..... /* Se puede poner tantos puntos como se quiera. */
Punto { -90, 60, 20 } /* Último punto del polígono. */
Abierto 0 /* Especifica si es abierto o no. */
Luminoso 0 /* Especifica si es luminoso o no. */
Material { ..... } /* Definición de su material. */
}

Esfera
{
Pos { -30, 20, 20 } /* Centro de la esfera. */

Radio 20/* Su radio. */
Luminoso 0 /* Especifica si es luminoso o no. */
Material { ..... } /* Definición de su material. */
}

Triangulo /* Definición de un objeto triángulo. */
{
Punto { -80, 60, 10 } /* Primer punto del triángulo. */
Punto { -80, 60, 10 } /* Segundo punto del triángulo. */
Punto { -90, 60, 20 } /* Tercer punto
        del triángulo. */
Abierto 0 /* Especifica si es abierto o no. */
Luminoso 0 /* Especifica si es luminoso o no. */
Material { ..... } /* Definición de su material. */
}

Plano /* Definición de un plano infinito. */
{
Pos { 0.1, 0.65, 0.2 } /* Vector normal al plano. */
Distancia 4 /* Distancia del plano al origen de coordenadas. */
Luminoso 0 /* Especifica si es luminoso o no. */
Material { ..... } /* Definición de su material. */
}

.....
```

Lenguaje de "programación de gráficos".

Saltarse los comentarios requiere un poco más de trabajo. Se define una variable llamada *comentario*, que indica el número de caracteres de demarcación se han leído. Por caracteres de demarcación se entiende los que señalan el comienzo y final de un comentario, es decir : "/", "*", "*" y "/".

La única regla a tener en cuenta es que los dos primeros y los dos últimos tienen que ir juntos.

Así, dentro del mismo bucle que salta los espacios, se comprueba si se ha encontrado el carácter "/". Si es así, se pone la variable contador a 1. La primera instrucción de la siguiente vuelta del bucle comprobará si el carácter es un "*", produciéndose en caso contrario un error de sintaxis. Si todo ha ido bien, la variable comentario toma el valor 2, hasta volver a encontrar un carácter "*". En este momento, pasará a valer 3, y se comprobará si el siguiente carácter es un "/". Si no es así, se decrementa la variable comentario, que vuelve a valer 2, y es como si se continuara dentro del comentario. En otro caso, el comentario ha terminado y la variable comentario vuelve a valer 0.

Hay que tener una última precaución, y es comprobar cada vez que se incrementa la variable índice si ha alcanzado el final del buffer, determinado por la variable *max_indice*. En este caso, se llama a la función *leer_buffer*, para comenzar con la siguiente ristra de caracteres.

FUNCIONES ESPECIALIZADAS

Para facilitar más el proceso de creación de objetos a partir de un lector, se han implementado varios métodos que intentan leer un tipo de dato concreto y devuelven el valor leído o un código de

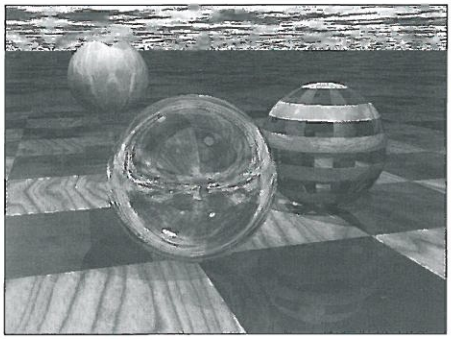


Figura 2 : Al aplicarle todos los recursos disponibles, la imagen se vuelve impresionante.

error para indicar que no se ha llevado a buen término. El proceso es muy similar en todos ellos, lo único que varía es el tipo de caracteres con los que trabaja cada uno. Estos procesos son :

- Leer un identificador. El método que lo implementa se llama *leer_ident*. Comienza llamando a la función *saltar_espacios*. El bucle es muy similar al de saltar espacios (sin tener en cuenta los comentarios). Dicho bucle continuará mientras se encuentre un carácter comprendido entre "A" y "Z", entre "0" y "9", o sea un punto u un subrayado. Para cada vuelta del bucle, se anotará el carácter leído y se incrementará un contador. Si dicho contador alcanza el valor de 255 caracteres, se produce un error de sintaxis. Como antes, también se comprueba en cada vuelta si se ha llegado al último carácter del buffer, en cuyo caso, se lee la siguiente ristra. Además, va actualizando las variables fila y columna. El proceso devuelve -1 (error) si se han leído más de 255 caracteres o si no se ha leído ninguno.
- Leer un número real. El método se llama *leer_real*, y es muy similar al anterior. Sólo admite números, el punto y el signo "-". Al igual que antes, si no logra leer ninguno correcto, devuelve -1.
- Leer un número entero. Se llama *leer_int*. Es idéntico al anterior, pero sólo con números.
- Leer una llave abierta, una llave cerrada y una coma. Los métodos se llaman, respectivamente, *leer_llave_ab*, *leer_llave_ce* y *leer_coma*. Tan sólo comprueban si, después de saltarse los espacios, se encuentra el carácter solicitado, y, como siempre, actualiza las variables fila y columna, y comprueba si tiene que volver a leer del fichero.

Con estas herramientas, ya se está en condiciones de poder leer todo el fichero con el formato especificado anteriormente.

ENTENDERLO TODO

Para poder leer el fichero entero, y siempre desde un punto de vista orientado a objetos, se ha implementado cada rutina de lectura dentro de cada objeto a leer. Nadie mejor que el objeto sabe qué campos han de leerse y si estos están correctos o no.

De esta forma, la rutina que lee el mundo, abre el fichero y comprueba qué elemento se está definiendo. Si, por ejemplo, es una esfera, llama a la rutina de lectura asociada a la esfera, quien, a su vez, comprueba si ha encontrado, por ejemplo, un material, llamado a la rutina implementada en el objeto material.

Para entenderlo más fácilmente, se van a estudiar todos los métodos desde el más sencillo al mas complejo.

¿ CÓMO SE LEE UN VECTOR ?

Un vector siempre estará formado por tres valores reales, separados entre sí por dos comas y todo ello encapsulado dentro de llaves. De esta forma, se ha diseñado un nuevo constructor, al cual se le pasa un puntero a un elemento lector.

Dicho constructor, comprueba si por este orden hay : una llave abierta, un real, una coma, un real, otra coma, otro real y una llave cerrada. Si encuentra todos estos campos, actualiza los campos del objeto que se está construyendo. Si no es así, muestra un mensaje de error de sintaxis y termina la ejecución del programa.

¿ CÓMO SE LEE UNA CÁMARA ?

La cámara es algo más compleja, ya que está formada por varios campos. Además, todos ellos son imprescindibles, así que también hay que comprobar que no falte ninguno. Por eso, se comienza definiendo cinco variables a 0, una por cada campo. A medida que se vayan leyendo los campos, estas variables pasan a valer 1, y al final se comprueban si todas tienen este valor. Si no es así, se informa de ello y termina el programa.

Lo primero de todo es comprobar si hay una llave abierta. Si es así, se hace un bucle que se repetirá hasta que la variable leer_ident devuelva un valor

BUSCAMOS A LOS MEJORES

- Programadores en C/C++
- Grafistas, diseñadores, dibujantes
- Expertos en lenguaje ensamblador
- Programadores 3D
- Expertos en Sonido
- Músicos con nociones de MOD, MIDI
- Programadores de juegos
- Animadores gráficos
- Infografistas con experiencia en 3D Studio, Photoshop, Deluxe Paint Animation
- Programadores con dominio de lenguajes multimedia: Authorware, Toolbook, Visual Basic, Macromedia Director, etc
- Expertos en comunicaciones

PARA DESARROLLAR

SOFTWARE MULTIMEDIA:

Presentaciones, libros interactivos, programas educativos, sistemas de comunicaciones, centros servidores de datos, videojuegos y mucho más...

SI ERES programador, músico o infografista y tienes ideas o proyectos en estudio de desarrollo ponte ya en contacto con nosotros.

TE OFRECEMOS las mejores condiciones y apoyo para producir tus programas y venderlos en el mercado nacional y extranjero. Formación en nuevas tecnologías y la mejor biblioteca de rutinas gráficas y sonido para desarrolladores.

Aportamos gráficos, rutinas o música, según las necesidades, para complementar cada proyecto.

Si estás interesado en unirte a una de las empresas más punteras en alta tecnología y desarrollo de software, no dejes pasar esta oportunidad. Envíanos carta con tus datos personales (curriculum vitae, con una muestra de tus anteriores trabajos) y un teléfono de contacto a:

DDM DIGITAL DREAMS MULTIMEDIA

DIGITAL DREAMS MULTIMEDIA
Ref. Programadores
C/ Vicente Muzas 15, 1º D - 28043 MADRID
Tf.: (91) 519.23.53 Fax (91) 413.55.77
BBS: (91) 519.75.75 Internet: ddm@servicom.es

erróneo. Para cada vuelta del bucle, se comprueba qué variable se está definiendo. Así, si se ha encontrado el identificador "DESDE", se está definiendo el punto el en que está situada la cámara, y, por lo tanto, tiene que venir a continuación un vector. Así que llama al constructor de vectores, pasándole el puntero al lector. Actúa de igual manera con el identificador "HASTA".

Para los identificadores "PIXELS_X" y "PIXELS_Y", lo que espera a continuación es un número entero, así que llama al método *leer_int*, definida en el objeto lector. Por último, el identificador "ANGULO_F" indica que se está definiendo el ángulo focal, así que intenta leer un real con el método *leer_real*.

Una vez comprobados todos los campos, comprueba si hay una llave de cerrar y llama al constructor de cámaras, pero ya con los campos obtenidos. Este método, devuelve los valores leídos de "PIXELS_X" y "PIXELS_Y", para poder generar posteriormente el fichero gráfico.

¿ Y UN MATERIAL ?

El constructor de material a partir de un lector es muy similar al anterior, con la diferencia de que se pueden ignorar todos los campos. O dicho de otra forma, basta con definir los campos que realmente se quieran modificar, quedando los demás con sus valores predeterminados. Los campos que se pueden encontrar son : "DIFUSA", que lee un vector a continuación ; "KR", "KS", "PS", "KT", "ESCALA_X" y "ESCALA_Y" que son acompañados por un número real ; "ESPEJO", que va con un entero y "COLOR" e "INTERSEC" que llevan otro identificador. Este identificador es el nombre del fichero de la textura que se está definiendo, y es el que se utiliza para llamar al constructor de texturas.

Termina comprobando si hay una llave cerrada y llamando al método *auxiliares*, para calcular las variables auxiliares necesarias.

¿ QUÉ SE HACE CON LOS OBJETOS ?

Los objetos que se definen en su mayoría son descendientes de la clase genérica objeto. Es decir, tienen los mismos campos y métodos que la clase objeto, y en algunos casos, varios campos y métodos más. Para seguir con esta misma filosofía dentro de la construcción de objetos a partir de un lector, La clase base hace la lectura de los campos base, y las clases descendientes comprueba si se está leyendo un campo de su clase o de alguna de sus ascendentes.

Por ello, hay un nuevo método virtual llamado *leer_campo*, al cual se le pasa el identificador y el lector y comprueba si es un campo de su clase, leyendo los datos auxiliares del mismo. Si no es así, llama a los métodos de las clases de las que desciende para comprobar si pertenece a ellas.

De esta forma, el constructor queda reducido a : leer una llave abierta, un bucle donde se llama a leer_campo y leer una llave cerrada. Si leer_campo devuelve un valor -1, es que dicho campo no pertenece ni a esa clase ni a sus antecesoras. El método leer_campo de la clase objeto, intenta leer los campos "POS", acompañado de un vector ; "MATERIAL", acompañado de la definición de un material y "LUMINOSO" y "ABIERTO", que van seguidos de un entero. No se obliga a que exista ninguno de ellos, por lo que basta con definir únicamente los que se vayan a usar.

Así, un descendiente que no tenga más campos que los anteriores no tiene que definir su constructor a partir de un objeto lector. Este es el caso de los objetos suelo y cielo, que heredan completamente el método de la clase objeto.

LA ESFERA ES POCO MÁS QUE UN OBJETO

Efectivamente, para leer una esfera basta con intentar leer un objeto y, además, un radio. Por lo tanto, se implementa el método virtual leer_campo, para que llame a su método heredado. Si este devuelve un error, intenta leer el campo "RADIO".

Una vez terminado el proceso, llama al constructor de esferas a partir de su posición y su radio.

LA FAMILIA DE LOS PLANOS

Leer un plano es exactamente igual que leer una esfera, solo que en lugar de intentar con el radio, intenta encontrar el campo "DISTANCIA". Por lo demás, es idéntico.

El caso del polígono es un poco más complicado, ya que tiene que permitir definir varios puntos. Como siempre, intentará leer algún campo perteneciente a sus clases antecesoras. Si no encuentra ninguno, comprueba si está definiendo un campo "PUNTO", en cuyo caso, lo almacena e incrementa el contador de puntos. Al acabar el proceso, comprueba si tiene al menos tres para poder generar el polígono. Si todo es correcto, llama al constructor de polígonos a partir de sus puntos.

El método que lee los triángulos es completamente igual al anterior, con la única salvedad de comprobar si se han leído exactamente tres puntos, ni uno más, ni uno menos.

Y, POR FIN, EL MUNDO

Una vez entendidos los métodos anteriores, el proceso de lectura del mundo es muy fácil de entender.

Primero, genera el lector pasándole el fichero que se quiere leer.

Luego, hace un bucle como en los anteriores objetos comprobando qué campo se está definiendo y llamando al constructor correspondiente. Estos campos pueden ser :

- "CAMARA", que llama al constructor de cámaras. Es obligatorio definirlo.
- "AMBIENTAL", que tan sólo es un vector.
- "FICHERO", que es el nombre con el que se va a generar la imagen. También es obligatorio definirlo.
- "CIELO" y "SUELO". Sobran las explicaciones.
- "POLIGONO", "ESFERA", "PLANO" y "TRIANGULO" : los objetos que se pueden definir.

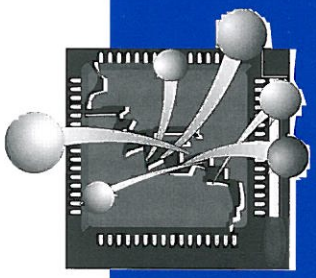
Al terminar el proceso, comprueba si están correctamente definidos el fichero y la cámara y, en caso afirmativo, llama al constructor de ficheros TGA, pasándole el nombre del fichero y las dimensiones obtenidas al leer la cámara.

BIBLIOGRAFÍA

- "An Introduction to Ray Tracing". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Graphics Gems". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Graphics Gems II". Editorial Academic Press.
- "Fundamentals of Three-Dimensional Computer Graphics". Alan Watt. Editorial Addison-Wesley.
- "Fractal Programming and Ray Tracing with C++". Roger T. Stevens. Editorial M & T Publishing.

INTRODUCCIÓN A VISUAL BASIC 4.0

Juan Manuel y Luis Martín



Con este artículo comienza un nuevo curso sobre la creación de aplicaciones Windows con Visual Basic 4.0, una de la herramientas de programación visual que más se ha popularizado en los últimos tiempos, y cuya última versión presenta novedades muy interesantes.

Desde hace varios años, se ha venido produciendo una importante expansión de los entornos gráficos, especialmente de Microsoft Windows. Esto ha supuesto para la comunidad de programadores la necesidad de adecuarse lo más rápidamente posible a una nueva forma de trabajo. A pesar de que los resultados son muy vistosos, las aplicaciones desarrolladas para este tipo de entornos con las herramientas tradicionales resulta bastante complejo, y requiere del programador un alto grado de conocimientos.

Sin embargo, la aparición en el mercado de las denominadas "Herramientas Visuales" ha facilitado mucho la labor del programador, permitiendo incorporar a programadores con unos conocimientos limitados al mundo de los entornos gráficos. Este tipo de herramientas unen a la sencillez de su manejo la rapidez en la obtención de resultados, así como la espectacularidad de los mismos, permitiendo crear en poco tiempo aplicaciones realmente profesionales.

Dentro de este grupo de herramientas, la que ha alcanzado un mayor nivel de popularidad y aceptación es Microsoft Visual Basic. Ello se debe principalmente a la rapidez con que permite crear el interfaz de la aplicación con el usuario, así como a la sencillez en la codificación de las instrucciones, ya que ésta se realiza en un lenguaje tan sencillo y popular como es Basic.

Aunque las primeras versiones de Visual Basic no obtuvieron el nivel de aceptación que cabía esperar, con la versión 3.0 se alcanzó un nivel de expansión tal, que le ha llevado a convertirse en un auténtico estándar dentro de la programación de aplicaciones para Windows. Este primer artículo de la serie se centrará principalmente en la filosofía utilizada por Visual Basic para la creación de aplicaciones Windows, proporcionando también una descripción de su entorno de desarrollo integrado.

NOVEDADES DE LA VERSIÓN 4.0

La principal novedad de esta última versión es que se incorporan las mejoras en el rendimiento y prestaciones de las aplicaciones de 32 bits, permitiendo así desarrollar aplicaciones para Windows'95, Windows 3.X y Windows NT. Para ello, se han incorporado nuevos controles específicos para dichos entornos.

Tanto los controles como el resto de herramientas, incluyen un completo soporte de la tecnología OLE 2.0, permitiendo incorporar a las aplicaciones la funcionalidad de los objetos creados con otras aplicaciones compatibles con dicho estándar. Además, es posible generar librerías de código con objetos OLE que también pueden ser reutilizados por otras aplicaciones. Para ello, se ha creado un nuevo tipo de módulos, los módulos de clases, en los pueden definirse objetos y especificar sus propiedades y métodos.

Se han incluido nuevos controles personalizados para el acceso a bases de datos, así como un nuevo control remoto que permite acceder fuentes de datos en el estándar ODBC 2.0. También, se ha incorporado el sistema de manejo de bases de datos Jet 3.0, permitiendo así desarrollar aplicaciones cliente/servidor.

El nuevo entorno de desarrollo integrado incluye un administrador de incorporación que permite añadir asistentes personalizados. De esta forma, es posible incorporar nuevas funciones al entorno de forma rápida y sencilla.

Visual Basic 4.0 se distribuye en tres ediciones distintas. La edición Estándar está pensada para usuarios noveles, permitiéndoles aprender las técnicas de desarrollo. La edición Profesional está pensada para programadores profesionales que desarrollan soluciones avanzadas. Finalmente, la edición Empresarial está pensada para equipos que deben generar y mantener grandes aplicaciones.

PROGRAMACIÓN GUIADA POR EVENTOS

La creación de aplicaciones Windows con Visual Basic supone una ruptura con los conceptos tradicionales de programación. Las aplicaciones creadas con lenguajes de programación *procedimentales* (Basic, Pascal, C, etc), consisten en programas cuyas líneas de código se ejecutan secuencialmente desde la primera. El flujo del programa sólo se ve alterado en determinados momentos previstos de antemano por el programador (estructuras de control, invocaciones de subprogramas, etc).

Por el contrario, las aplicaciones creadas con Visual Basic están basadas en otra filosofía, denominada *Programación Guiada por Eventos*. Aunque el código necesario para el funcionamiento de las aplicaciones se escribe en Basic, los conceptos manejados son nuevos y se encuentran más cercanos a las técnicas de Programación Orientada a Objetos que a las tradicionales. Así, se requiere que el programador piense en términos de formularios, objetos, controles, eventos, métodos, etc.

Gran parte del trabajo de diseño de las aplicaciones se realiza de forma visual, mediante movimientos y pulsaciones del ratón. El programador debe *dibujar* controles sobre ventanas vacías, denominadas formularios, que actuarán de soporte de los mismos. Los formularios, junto con sus controles, constituirán el interfaz de la aplicación con el usuario. Además, los distintos elementos que se van añadiendo a las aplicaciones pueden ser probados y depurados continuamente, sin necesidad de abandonar el entorno de desarrollo.

La principal diferencia de Visual Basic frente a otros lenguajes de programación es que se trata de un lenguaje de programación guiado por eventos. Esto quiere decir que las instrucciones que componen el código del programa sólo se ejecutan cuando ocurren ciertos sucesos, denominados *eventos*. Estos eventos son susceptibles de ser detectados por los distintos elementos u objetos que componen las aplicaciones (formularios, controles, etc). Así, dependiendo del tipo de evento ocurrido y del objeto que lo ha detectado, se ejecutará una u otra sección del código.

Para crear una aplicación con Visual Basic es necesario crear un Proyecto

Los eventos pueden ser producidos por diferentes fuentes:

- Por el usuario: Cuando éste realiza un clic con el ratón, cuando pulsa una tecla, etc.
- Por el ordenador: Cuando se produce un aviso del temporizador del sistema, un error de ejecución, etc.
- Por la aplicación: Cuando una instrucción invoca directamente un evento.
- Por otras aplicaciones: Cuando se produce un intercambio de datos con otras aplicaciones *Windows* que se están ejecutando concurrentemente.

Las secciones del código que se ejecutan en función del evento ocurrido y del objeto que lo ha detectado son pequeños subprogramas denominados *procedimientos de evento*. Estos procedimientos deben ser escritos en Basic por el programador, y deben contemplar las acciones que deben realizarse como respuesta a dicho evento. Así, por ejemplo, cuando se realiza un clic sobre un botón de comando, llamado *Command1*, se ejecutará el procedimiento de evento *Command1_Click()*.

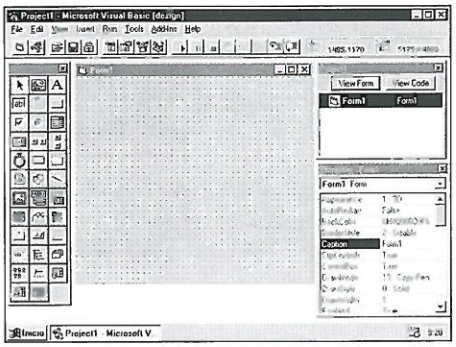


FIGURA 1. El Entorno de Desarrollo de Visual Basic 4.0.

Los distintos objetos de que dispone Visual Basic para la creación de aplicaciones poseen un conjunto de procedimientos de evento para responder a un cierto repertorio de eventos. El programador deberá decidir a qué eventos responderá cada objeto y escribir el código necesario en el procedimiento de evento correspondiente.

Para facilitar la tarea de programación, *Visual Basic* suministra la primera y última línea de cada procedimiento de evento. El espacio existente entre estas dos líneas será el que deba utilizarse para escribir el código a ejecutar. Si no se desea que el objeto responda al evento, debe dejarse en blanco el espacio entre ambas líneas. Así, por ejemplo, si no se desea que el botón de comando *Command1* responda cuando el usuario hace *clic* sobre él, el procedimiento de evento correspondiente quedará de la siguiente forma:

```
Private Sub Command1_Click ()
End Sub
```

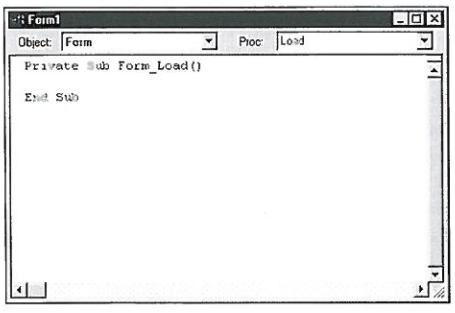


FIGURA 2. La Ventana de Código.

OBJETOS, FORMULARIOS Y CONTROLES

Las aplicaciones realizadas con Visual Basic se componen de *objetos*. Los objetos pueden definirse como combinaciones de código y datos que pueden ser tratadas como una sola unidad. Por tanto, su manejo se realiza como unidades independientes. Los objetos son las estructuras básicas que componen el núcleo de las modernas técnicas de programación.

Los principales objetos que se utilizan para la creación de aplicaciones con Visual Basic son los siguientes:

- *Formularios (Form):* Representan las distintas ventanas de que constará la aplicación.
- *Controles (Controls):* Son los elementos gráficos que deben incluirse en cada formulario para permitir la interacción entre la aplicación y el usuario.

Los formularios, junto con los controles que contienen, forman el interfaz de la aplicación con el usuario. Visual Basic dispone de un amplio repertorio de controles predefinidos, que permiten realizar diversas funciones. Ejemplos de controles son los botones de comando, las cajas de texto, las listas, etc. Cada uno de ellos está representado por un icono diferente en la Caja de Herramientas.

Los objetos, ya sean formularios o controles, poseen una serie de atributos, denominados *propiedades* (*properties*), que definen tanto su aspecto como su comportamiento. Cada una de estas propiedades posee un nombre y un valor que permite indicar características del objeto tales como el color, el tamaño, la posición, etc. Estos valores se establecen en la fase de diseño de la aplicación, aunque también pueden ser modificados durante la ejecución de la misma.

Como ya se mencionó anteriormente, cada tipo de objeto (formulario, botón de comando, caja de texto, etc) posee también un repertorio de procedimientos de evento, que le permiten responder a los eventos que es capaz de detectar. El programador debe seleccionar aquellos eventos a los que debe responder el objeto y escribir el código necesario en su procedimiento de evento correspondiente.

Finalmente, los objetos poseen una serie de procedimientos propios, llamados *métodos* (*methods*), que permiten manipularlos. Existen métodos para mover el objeto, para dibujar en su interior, etc. Para invocar un método en una sentencia es necesario escribir el nombre del objeto y del método, separados por un punto. Así, por ejemplo, para escribir un texto sobre el formulario *Form1* puede utilizarse el método *Print* de la siguiente forma:

Form1.Print "Bienvenido a Visual Basic"

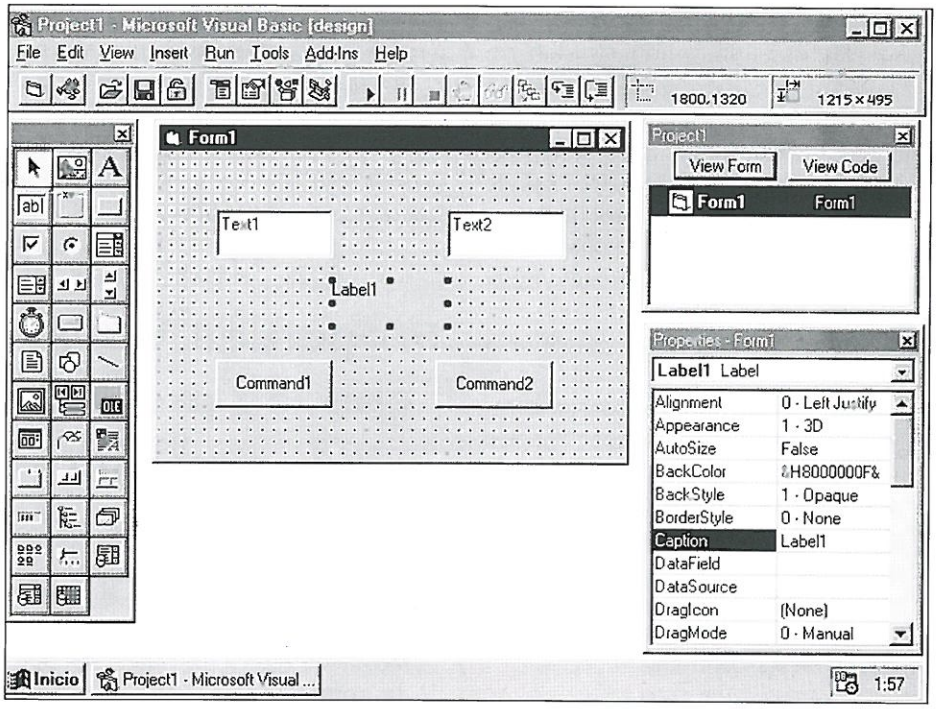


FIGURA 3. Situación de los controles en el formulario.

Gran parte del trabajo de diseño de las aplicaciones se realiza de forma visual, mediante movimientos y pulsaciones del ratón

PROYECTOS Y MÓDULOS

Para crear una aplicación con Visual Basic es necesario crear un *Proyecto*. Este proyecto contendrá todas las especificaciones necesarias para el funcionamiento de la aplicación. En esencia, los proyectos de Visual Basic están constituidos por un conjunto de archivos, denominados *módulos*, que contienen las especificaciones de cada uno de los elementos que intervienen en la creación de la aplicación.

Cada proyecto se almacena en un archivo de extensión .VBP que contiene la lista de los archivos individuales que componen la aplicación. El archivo de proyecto también guarda las condiciones del entorno de desarrollo la última vez que éste se salvó.

Cada formulario se almacena en un archivo individual de extensión .FRM. Este tipo de archivos se denominan *módulos de formulario*, y contienen las propiedades, métodos y procedimientos de evento del formulario y de todos los controles que contiene. Visual Basic también permite incluir en los proyectos otros archivos de extensión .BAS, que solamente contienen declaraciones y procedimientos. Este tipo de archivos se les denomina *módulos estándar*. Los procedimientos que contienen se denominan *procedimien-*

tos generales, con el fin de diferenciarlos de los procedimientos de evento.

Los proyectos también pueden incluir otros tipos de archivos, como módulos de clases, controles a medida y archivos de recursos, que irán siendo analizados en posteriores artículos.

En resumen, el diseño de una aplicación con Visual Basic debe contemplar los siguientes pasos:

- Creación del proyecto asociado a la aplicación.
- Diseño de los formularios que componen la aplicación, y especificación de sus propiedades.
- Inserción de los controles de cada formulario, y especificación de sus propiedades.
- Diseño de los procedimientos de evento, tanto de los formularios como de los controles.

EL ENTORNO DE DESARROLLO

Cuando se ejecuta la edición profesional de Visual Basic 4.0 se despliega su *Entorno de Desarrollo Integrado* (IDE), obteniéndose una pantalla similar a la mostrada en la figura 1. Como puede apreciarse, está compuesto por diversas ventanas que se visualizan sobre el escritorio de Windows, más otras que no se visualizan inicialmente.

Aunque el manejo del entorno puede parecer complicado, dado el elevado número de elementos que lo componen, es posible familiarizarse con él en poco tiempo. Cada una las ventanas tiene una misión específica en la creación de aplicaciones:

- La ventana principal o *Main Window* es el soporte del sistema de menús de Visual Basic. Cada uno de los menús contiene diversas opciones o comandos, permitiendo así manejar archivos, editar código, ejecutar y depurar aplicaciones, etc.

Sin embargo, también contiene la *Barra de Herramientas* (*Toolbar*), que proporciona un acceso rápido a las operaciones más frecuentes. La tabla 1 muestra el nombre y la función que realiza cada uno de sus botones.

- La caja de herramientas o *Toolbox* contiene una representación en forma de icono de todos los tipos de controles y objetos insertables que pueden ser dibujados sobre el formulario (botones de comando, cajas de texto, listas, barras de desplazamiento, etc).

Para insertar un control en un formulario debe hacerse clic sobre su icono correspondiente en la caja de herramientas, y dibujarlo sobre el formulario como en cualquier aplicación Windows de dibujo.

- La Ventana de Propiedades o *Properties Window* contiene una lista con las propiedades del objeto que encuentra seleccionado, así como el valor actual de cada una. Para acceder a ella, puede seleccionarse la opción *Properties* en el menú *View*, hacer clic en el botón *Properties* de la barra de herramientas o pulsar [F4].

Contiene una Lista de Objetos, indicando el nombre del objeto actualmente seleccionado. Cuando se hace clic sobre ella, se despliega una lista con los objetos del formulario actual, permitiendo seleccionar otro. También contiene la Lista de Propiedades, con los nombres de las propiedades del objeto seleccionado.

La Ventana de Propiedades o Properties Window contiene una lista con las propiedades del objeto que encuentra seleccionado

A la izquierda se visualiza el nombre de la propiedad, mientras que a la derecha aparece el valor actual.

Para cambiar una propiedad basta seleccionarla, haciendo clic sobre ella, y teclear el nuevo valor. Si el repertorio de valores es limitado se desplegará una lista, permitiendo seleccionar uno de los posible valores.

- La Ventana de Proyecto o *Project Window* contiene los nombres de todos los ficheros que forman parte del proyecto. También permite acceder a las ventanas de formulario y código de los módulos correspondientes. Para acceder a ella, puede seleccionarse la opción *Project* en el menú *View*, hacer clic en el botón *Project* de la barra de herramientas o pulsar [Ctrl-R].

Al hacer clic sobre un módulo, éste queda seleccionado. Para acceder al formulario de hacerse clic en el botón *View Form*, mientras que para acceder al código debe hacerse clic en el botón *View Code*.

- La Ventana de Formulario o *Form Window* es el soporte para la creación del interfaz de la aplicación con el usuario. Sobre ella se irán dibujando los distintos controles. Existirá una ventana de formulario para cada módulo de formulario del proyecto.

En esta ventana, el control que se encuentra seleccionado se visualiza con unas marcas. La ventana de propiedades hará referencia a este control.

- La Ventana de Código o *Code Window* contiene el código de

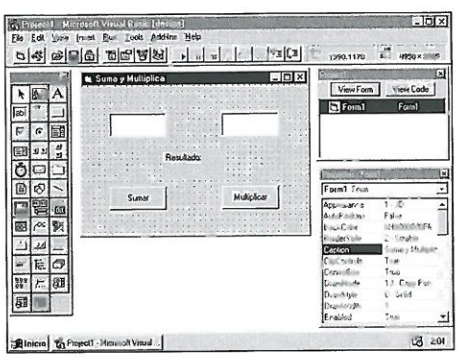


FIGURA 4. Cambio de la propiedad Caption de los controles

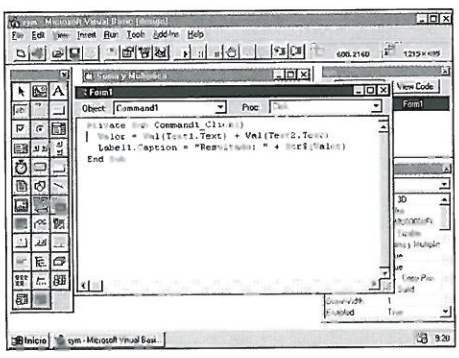


FIGURA 5. Escritura de los procedimientos de evento.

todos los procedimientos de evento y generales que forman parte del proyecto, permitiendo su edición. Inicialmente no se encuentra visible, pero puede accederse a ella seleccionando la opción *Code* en el menú *View*, haciendo clic en el botón *Code* de la barra de herramientas o pulsando [F7]. También es posible acceder a ella haciendo doble clic sobre cualquier objeto de la ventana de formulario. Existirá una ventana de código para cada módulo de formulario, módulo estándar o módulo de clase del proyecto.

La figura 2 muestra el aspecto de esta ventana. Como puede apreciarse, dispone de dos lista desplegables, una que permite seleccionar el objeto (*Object*) y otra el procedimiento (*Proc*) que se desea editar. Al seleccionar un objeto y un evento se visualiza el código del procedimiento de evento correspondiente, permitiendo su edición. En el caso de la figura se visualiza el procedimiento de evento *Load* del objeto *Form*.

TABLA 1

Nombre	Descripción
Form	Crea un nuevo formulario y lo añade al proyecto
Module	Crea un nuevo módulo estándar y lo añade al proyecto
Open Project	Abre un proyecto ya existente
Save Project	Salva el proyecto actual
Lock Controls	Bloquea o desbloquea los controles del formulario actual
Menu Editor	Visualiza el editor de menús
Properties	Visualiza la ventana de propiedades
ObjectBrowser	Visualiza la ventana del hojeador
Project	Visualiza la ventana de proyecto
Start	Inicia la ejecución de la aplicación
Break	Realiza una pausa en la ejecución (modo *Break*)
End	Detiene la ejecución de la aplicación
Toggle Braeakpoint	Inserta un punto de parada en la línea actual
Instant Watch	Visualiza el valor de la selección actual en la ventana de código
Calls	Visualiza la estructura de llamadas a procedimiento activas
Step Into	Ejecuta paso a paso la siguiente instrucción entrando en los procedimientos
Step Over	Ejecuta paso a paso la siguiente instrucción sin entrar en los procedimientos

Botones de la barra de herramientas.

CREACIÓN DE UNA APLICACIÓN SENCILLA

La mejor forma de comprender las distintas herramientas que componen el entorno de desarrollo de Visual Basic consiste en crear una pequeña aplicación ejemplo. Se trata de una ventana que permite realizar la suma o el producto de dos números tecleados por el usuario. Los pasos a seguir son los siguientes:

1. Crear un nuevo proyecto, seleccionando la opción *New Project* en el menú *File*.

2. Añadir al formulario una caja de texto (*Text1*), haciendo doble clic sobre su icono representativo en la caja de Herramientas. Arrastrar el control dentro del formulario hasta la posición deseada, usando como referencia la figura 3.

3. Repetir el pasos 2 hasta obtener dos cajas de texto (*Text1* y *Text2*), una etiqueta (*Label1*) y dos botones de comando (*Command1* y *Command2*). Situarlos aproximadamente como se indica en la figura 3.

4. Seleccionar el formulario, haciendo clic sobre él, con el fin de cambiar su título. Acceder a la ventana de propiedades, pulsando [F4].

5. Seleccionar la propiedad *Caption*, haciendo clic sobre ella, y teclear el texto "Suma y Multiplica". El cambio quedará reflejado automáticamente en la barra de título del formulario.

6. Repetir los pasos 4 y 5 para cambiar el título (propiedad *Caption*) de los controles, según se muestra en la figura 4.

7. Hacer doble clic sobre el botón *Command1* para desplegar la ventana de código. En ella se visualizará el procedimiento de evento *Click* del control, tal y como se muestra en la figura 5.

8. Para que al hacer clic sobre el botón *Command1* se visualice la suma en *Label1*, debe teclearse el siguiente código:

```
Private Sub Command1_Click ()
Valor = Val(Text1.text) + Val(Text2.Text)
Label1.Caption = "Resultado: " + Str$(Valor)
End Sub
```

9. Repetir los pasos 7 y 8 para *Command2*, escribiendo el siguiente código:

```
Private Sub Command2_Click ()
Valor = Val(Text1.text) * Val(Text2.Text)
Label1.Caption = "Resultado: " + Str$(Valor)
End Sub
```

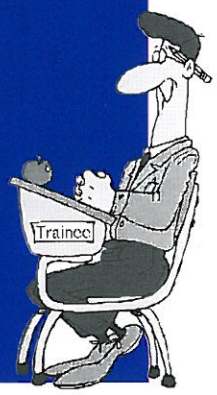
10. Pulsar [Ctrl+S] para salvar el módulo de formulario con el nombre SYM.FRM y el proyecto con SYM.VBP.

11. Para ejecutar y probar la aplicación bastará con pulsar la tecla [F5]. Para cerrarla bastará con hacer clic sobre el botón de cierre de la ventana.

12. Para crear un archivo ejecutable debe seleccionarse la opción *Make EXE File* en el menú *File*, indicando SYS.EXE como nombre del archivo.

CONCEPTOS BÁSICOS SOBRE WIN32

José C. Remiro



Conjuntamente con el lanzamiento de Windows NT, Microsoft presentó un nuevo API encargado de programar dicho sistema operativo, Win32, y que como posteriormente se ha visto, ha ido sustituyendo a su predecesor Win16 en Windows 95 y se alza como el API base para sus futuros sistemas operativos de 32 bits.

En el anterior artículo hablamos de las capacidades de Visual C++, de la bondad y sencillez de utilizar su librería de clases MFC 4.0 para programación de Windows, y tratamos algunas de las utilidades para la automatización y reutilización de código, pero hagamos una reflexión, ¿No tiene sentido conocer aunque sea mínimamente el sistema sobre el cual trabajan estas herramientas? Está pregunta puede tener varias soluciones, ya que si vamos a desarrollar aplicaciones sencillas sin tener que utilizar funcionalidades especiales del sistema, probablemente no sea necesario bajar de la interfaz suministrada por las MFCs, pero si vamos a desarrollar todo tipo de aplicaciones, más o menos complicadas, que requieran elementos no incluidos en las MFCs, vendrá muy bien un conocimiento del sistema y del modo quizá más tradicional aunque menos rápido de desarrollar para Windows, la programación en C del API Win32.

Windows 95 vs Windows 3.1, o 32 bits vs 16 bits

Como todos sabemos Microsoft nos presenta a Windows 95 como un sistema operativo de 32 bits frente a Windows 3.x que era un entorno de 16 bits, y digo entorno y no sistema operativo, ya que Windows 3.x necesita de DOS para poder funcionar. Pero la pregunta es ¿Qué quiere decir la afirmación Windows 95 es un sistema operativo de 32 bits? La solución a esta pregunta pasa por conocer aunque sea mínimamente la arquitectura de los microprocesadores de Intel 80386 y superiores.

A muy grosso modo podríamos decir que este apellido de 32 bits parte de lo siguiente. En el 80386 se implementan una serie de novedades encaminadas a permitir la multitarea real entre diferentes aplicaciones como son la memoria virtual y paginación, la protección entre tareas y dentro de cada una de ellas, instrucciones para permitir conmutar cada tarea, etc. Una de las novedades en esta familia es que se disponen de segmentos con un espacio de direccionamiento de 32 bits, permitiendo eliminar la barrera que existía anteriormente y conocida por los programadores de DOS, que limitaba cada segmento a 64 Kbytes, es decir, un espacio de direccionamiento de 16 bits. Windows 3.1 al tener que ser compatible con los sistemas basados en microprocesadores 80286, no implementa muchas de las funcionalidades suministradadas por los 80386, manteniendo la limitación de 64K por segmento que tantos dolores de cabeza ha dado a muchos de nosotros. Otra limitación es el método de realizar la multitarea, ya que en Windows 3.1 es cooperativa, es decir, no se devuelve el control a otra tarea hasta que no se cede voluntariamente el control al sistema. En Windows 95, al no tener que ser compatible con los sistemas 80286, se han utilizado las propiedades basadas en los microprocesadores 80386 y superiores, permitiendo entre otras cosas, tener lo que se conoce por multitarea real, donde es el propio sistema el cual transfiere el control a cada tarea sin esperar a que cada una devuelva el control al sistema. Además permite tener espacios de direccionamiento protegidos de 32 bits por cada tarea o proceso,

saltando así una de las mayores limitaciones de Windows 3.x.

Todos estos cambios desde una versión a otra, van encaminados a la mejora del rendimiento y robustez utilizando las propiedades de los microprocesadores 80386. Windows 95 utiliza el API de 32 bits, al igual que lo hace Windows NT y que es ligeramente incompatible con el API de 16 bits de Windows 3.x.

¿Qué es el API Win32?

El API Win32 es básicamente una librería de funciones que permiten al programador utilizar todos los recursos y propiedades del sistema operativo Windows para realizar sus propias aplicaciones sobre éste. El API nos brinda la posibilidad de realizar aplicaciones totalmente independientes del hardware de los diferentes ordenadores donde sea ejecutado y liberándonos de muchas tareas de programación. El API Win32 nos permite implementar mediante sencillas llamadas a funciones, aplicaciones con una interfaz gráfica realmente espectacular o con un sistema de menús realmente bueno o crear un editor de texto de forma casi instantánea, pero es además y debido a la cantidad de funciones incluidas, un poco difícil de utilizar para los que empiezan a manejarlo, ya que se cuentan las estructuras, funciones y macros por varios cientos e incluso miles.

Como antes hemos dicho Win32 surge junto a Windows NT y lleva

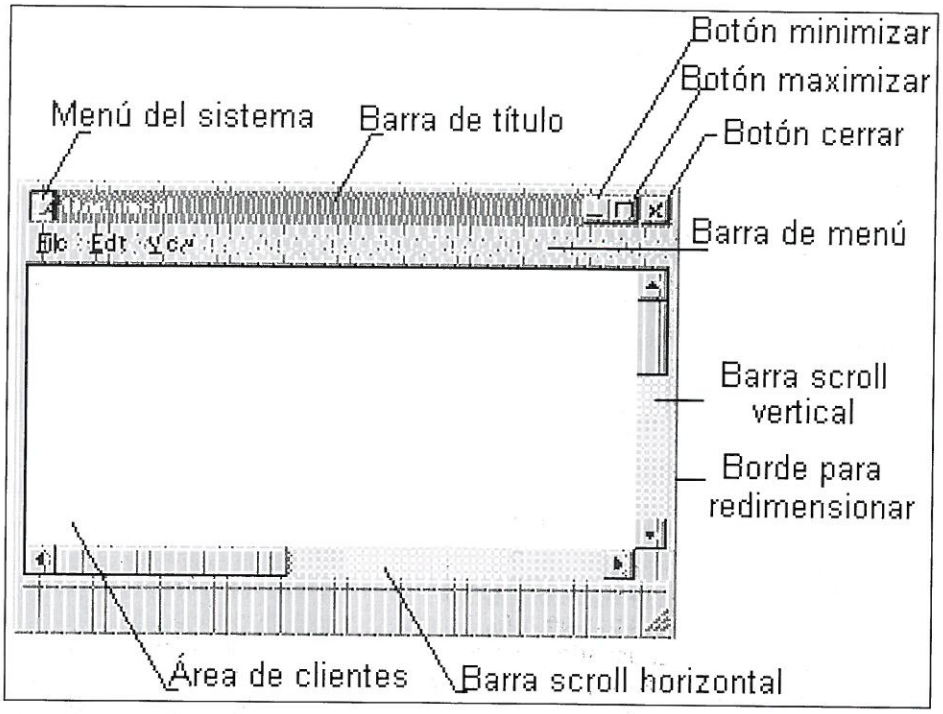


Figura A. Ventana principal típica y sus componentes.

Microsoft nos presenta a Windows 95 como un sistema operativo de 32 bits frente a Windows 3.x que era un entorno de 16 bits

muchas funciones necesarias para programar este sistema operativo, además hemos indicado que Windows 95 también utiliza Win32, pero entonces surge la pregunta, ¿Tiene Windows 95 todas las funcionalidades de Windows NT? Absolutamente NO, pero entonces ¿Qué ocurre cuando llamamos a una función del API Win32 que sólo está soportado por Windows NT? Pues la respuesta es quizá la más sencilla posible, simplemente se devuelve un código de error indicando que no está soportado. Esto nos lleva a afirmar que cuando realicemos algún tipo de llamada a una función del API, debemos comprobar los posibles valores de retorno para conocer si se ha producido algún tipo de error y no presuponer erróneamente que las funciones API siempre funcionan correctamente.

Existen varios puntos sobre al API que debemos tener suficientemente claros como son ¿Qué son las ventanas y sus tipos?,¿Qué son los mensajes y las colas de mensajes?, ¿Las funciones de gestión de mensajes? o ¿Qué es el GDI?. Además existen muchos otros puntos que junto con los expuestos ahora iremos detallando a lo largo del curso.

Handles

Si hay un concepto que fluye a lo largo de toda la programación para Windows es el *handle*. Un *handle* no es más que un identificador que permite localizar las diferentes ventanas, instancias, objetos del GDI,... Se puede definir como una especie de apuntador que permite tener acceso a lo apuntado independientemente de los cambios de posición en memoria que realiza Windows internamente. Todo o casi todo en Windows se localiza o controla mediante su *handle*, el cual normalmente es devuelto al crear cada ventana, instancia o objeto GDI,... y que utilizaremos sin preocuparnos por su valor.

Instancias

Windows al ser un sistema multitarea, permite abrir varias copias de una misma aplicación al mismo tiempo, esto implica que debe existir algún método para diferenciar la ejecución entre todas las aplicaciones del mismo tipo y minimizar el gasto de memoria compartiendo el segmento código ya que es invariable e idéntico para todas y definiendo un segmento de datos particular para cada aplicación. Para diferenciar entre las varias aplicaciones se crean unos handles a cada segmento de datos, llamados instancias.

¿Qué son las ventanas?

La pregunta del millón es, ¿Qué importancia pueden tener las ventanas

dentro de un entorno llamado "Windows"? Parece obvio que toda. Cualquier aplicación para el entorno Windows crea al menos una ventana, la ventana principal, que sirve como interfaz entre el usuario y la aplicación. Cada ventana puede tener entre otros elementos, una barra de título, una barra de menú, un menú de sistema, un botón para minimizar, otro para maximizar, un borde que permita redimensionar la misma, el área de cliente y las barras de *scroll*. **(Ver Figura A).**

Además de la ventana principal existen otros tipos de ventanas como son los controles, las cajas de diálogo y las cajas de mensajes. Los controles son un tipo de ventanas especializadas, encargadas de obtener de forma directa información del usuario, ya sea en forma de pulsar un botón, en forma de elegir una opción, de insertar un texto,... Las cajas de diálogo son ventanas con uno o varios controles y encargadas de mostrar y obtener también información de los usuarios por medio de sus diferentes controles. El otro tipo de ventanas especializadas son las cajas de mensajes, que se encargan de mostrar un nota, mensaje o alarma relacionada con algún tipo de acción tomada por el usuario. **(Ver Figura B).**

El proceso de creación de una ventana reside sobre varias funciones como son ***CreateWindow*** o ***CreateWindowEx*** para ventanas de tipo general, ***CreateDialog*** y ***DialogBox*** para cajas de diálogo y ***MessageBox*** para cajas de mensajes. Antes de crear una ventana mediante *CreateWindow* o *CreateWindowEx*, debemos registrar el tipo de ventana que queremos crear o bien utilizar una ventana ya registrada. Registrar una ventana implica definir una serie de atributos, así como asignar una función de gestión de mensajes que compartirán todas aquellas ventanas creadas de esta clase.

Mensajes y colas de mensajes

Al contrario de los que ocurre en las aplicaciones escritas para sistemas operativos como MS-DOS, no existen llamadas explícitas para solicitar la entrada de datos al usuario, sino que

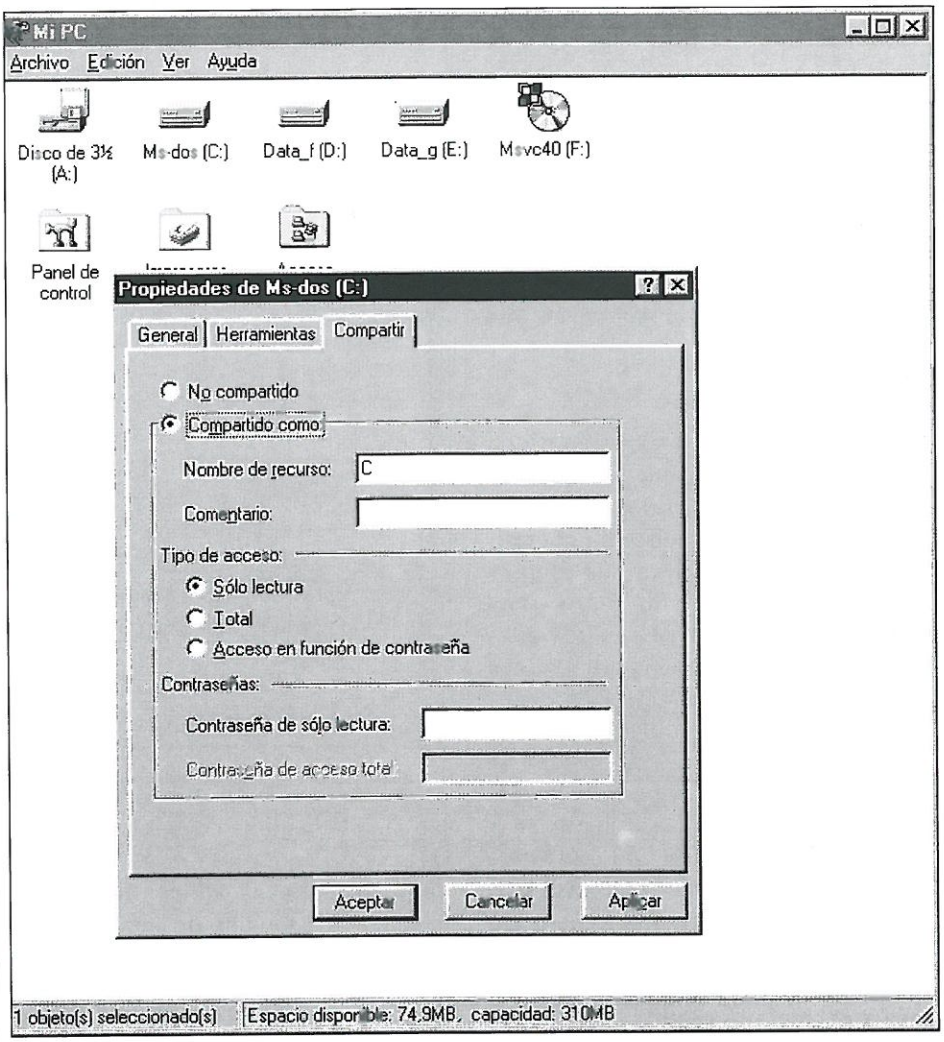


Figura B. Diferentes tipos de ventanas.

cualquier tipo de entrada del usuario o cambio en el sistema, envía un mensaje a la ventana que corresponda. Cada ventana tiene asociada una función para atender todos los mensajes que reciba y poder tomar las acciones pertinentes.

El formato de los mensajes consta de un handle a la ventana a la cual va dirigido el mensaje, un identificador de mensaje que no es más que una constante que indica el número de mensaje y dos parámetros de 32 bits donde se envían los diferentes datos asociados a cada mensaje.

Existen dos funciones para enviar un mensaje a una ventana determinada, *PostMessage* y *SendMessage*. La primera implica que el mensaje se envía y se almacena en una cola de mensajes que tiene cada aplicación, donde se procesan los mensajes según el orden de llegada. Después de almacenar el mensaje en la cola se devuelve el control a la aplicación que envío el mensaje permitiendo continuar la ejecución. SendMessage implica enviar el mensaje directamente a la función de gestión de mensajes de la ventana, no devolviéndose el control hasta que no se termina de procesar el mensaje completamente dentro de dicha función.

Cada aplicación tiene el deber de obtener los mensajes de su cola de mensajes, así como de procesarlos cuando sea pertinente. Esta operación se realiza mediante el llamado bucle de mensajes, que puede incluir las siguientes funciones: ***GetMessage***, ***TranslateMessage*** y ***DispatchMessage***. ***GetMessage*** se encarga de obtener los mensajes de la cola, **TranslateMessage** se ocupa de traducir las pulsaciones sobre una tecla en el carácter adecuado, y

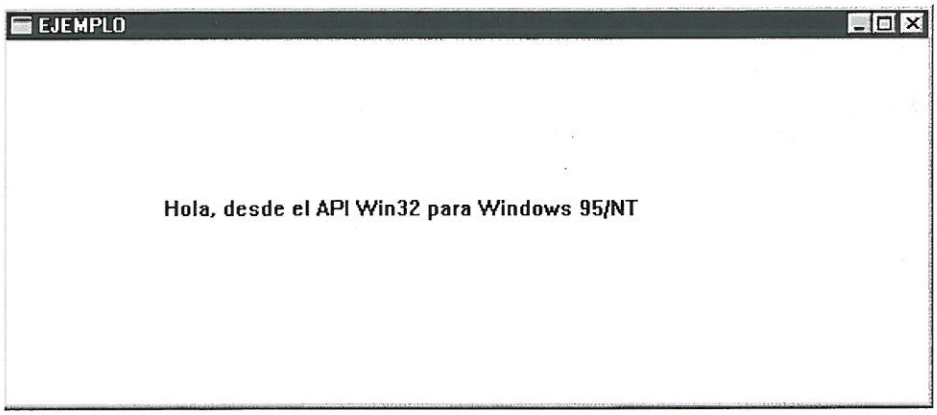


Figura C. Aplicación mínima para Windows.

DispatchMessage se encarga de enviar el mensaje a la función de gestión de mensajes de la ventana. Al estar estas tres funciones en un bucle infinito, se está observando constantemente los mensajes recibidos.

Funciones de gestión de mensajes

Como ya hemos indicado anteriormente cada clase de ventana tiene asociado una función para gestionar los mensajes recibidos por cada ventana de esa clase. El formato genérico de estas funciones es el siguiente:

```
LRESULT WINAPI <NombreFuncion>-
(HWND hWnd, UINT uMsg, WPARAM
wParam, LPARAM lParam);
```

donde,

hWnd es el handle de la ventana sobre la cual se dirige el mensaje.

uMsg es el identificador del mensaje.
wParam y *lParam* son los dos parámetros, enteros de 32 bits.

La función devuelve *LRESULT*, que no es más que un entero de 32 bits, como resultado a las acciones tomadas.

Dentro de estas funciones se debe tratar cada mensaje en función de su identificador, observar los parámetros recibidos y ejecutar las acciones pertinentes. Si no vamos a procesar algún mensaje se debe reenviar llamando a la función ***DefWindowProc*** pasándole los mismos cuatro campos recibidos en nuestra función.

El primer programa para Windows

Una vez visto de manera mínima los componentes básicos de una aplicación para Windows, vamos a mostrar un sencillo programa de ejemplo donde se ilustra el modo de enlazar todo lo visto.

El punto de entrada en una aplicación Windows ha sido siempre *WinMain*, y aunque en Win32 podemos definir otro nombre, no hay tampoco razón para cambiar, por tanto el formato es:

```
int APIENTRY WinMain(HINSTANCE
hInstance,
                     HINSTANCE hPrevInst,
                     LPSTR lpCmdLine,
                     int nCmdShow)
```

donde,
hInstance es un handle a la instancia de la aplicación.
hPrevInst en Win32 es siempre NULL. Anteriormente identificaba la existencia de otra aplicación de este tipo abierta con anterioridad.
lpCmdLine es una cadena terminada en NULL que contiene la línea de comandos.
nCmdShow es el modo con el cual debe ser mostrada la ventana.

El API Win32 es básicamente una librería de funciones que permiten al programador utilizar todos los recursos y propiedades del sistema operativo

Dentro de *WinMain* se debe registrar la clase de ventana a utilizar, definiendo el estilo de la ventana, la función de gestión de mensajes, tamaño de los datos extra de la clase y de la ventana (normalmente 0), la instancia de la aplicación iniciada, el icono por defecto, el cursor por defecto, el color de fondo por defecto, el menú (si es NULL no tiene menú) y el nombre de la clase.

```
wndclass.style = CS_HREDRAW |
CS_VREDRAW;
wndclass.lpfnWndProc = (WND-
PROC)WndProc;
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = hInstance;
wndclass.hIcon = LoadIcon (NULL,
IDI_APPLICATION);
wndclass.hCursor = LoadCursor
(NULL, IDC_ARROW);
wndclass.hbrBackground =
GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName = NULL;
wndclass.lpszClassName = "EJEM-
PLO";

if (!RegisterClass (&wndclass))
{
return FALSE;
}
```

La función *RegisterClass* se encarga de registrar la clase de ventana mediante la estructura *WNDCLASS* llenada con anterioridad. Posteriormente a esto podemos crear ya la ventana mediante la función ***CreateWindow***, con el siguiente formato:

```
hwndMain = CreateWindow ("EJEM-
PLO", "EJEMPLO",
      WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0,
      CW_USEDEFAULT, 0,
      NULL, NULL, hInstance, NULL);
```

Esta función recibe el nombre de la

clase de ventana que queremos crear, el título, el tipo de ventana, la posición x, la posición y, el ancho, el alto, el handle a la ventana padre (NULL si es la ventana principal y no tiene padre), un handle al menú (NULL si no tiene), la instancia y el parámetro enviado junto al mensaje *WM_CREATE*, enviado al crear la ventana. Después de haber creado la ventana y por tanto tener un handle válido a ella, es decir, distinto de NULL, podemos mostrarla llamando a la función ***ShowWindow***.

```
if (!hwndMain)
{
  return FALSE;
}

ShowWindow (hwndMain, nCmdShow);
```

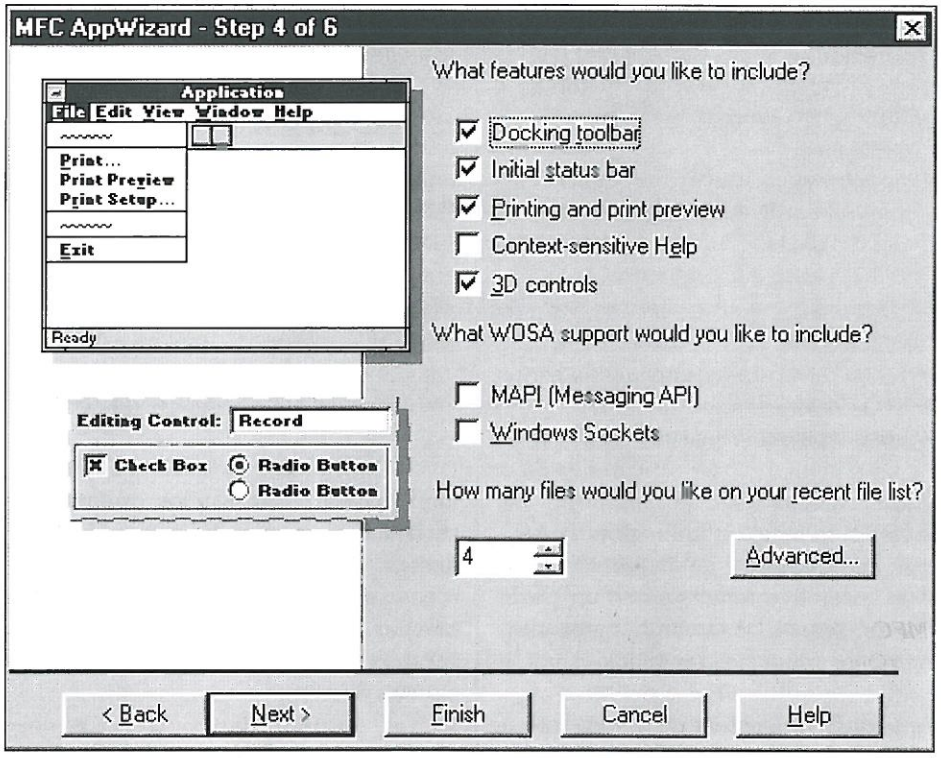
Después de todo esto, podemos crear el bucle de mensajes ya que la ventana ya está en disposición de recibir mensajes. Para ello implementamos un bucle while donde se incluye una llamada a la función *GetMessage*, la cual devuelve TRUE mientras el mensaje sea distinto de WM_QUIT (mensaje enviado para cerrar la aplicación) y FALSE si se recibe dicho mensaje.

```
while (GetMessage (&msg, NULL, 0, 0))
{
  TranslateMessage (&msg);
  DispatchMessage (&msg);
}
```

Dentro del bucle de mensajes se debe llamar a las funciones *TranslateMessage* para traducir las pulsaciones de teclas a mensajes *WM_CHAR*, y a la función *DispatchMessage* encargada de enviar el mensaje leído a la función de gestión de mensajes.

Por último se la función de gestión de mensajes tendría un formato como el que sigue:

```
LRESULT APIENTRY WndProc(HWND hWnd, UINT uMsg,
WPARAM wParam, LPARAM lParam)
{
  PAINTSTRUCT ps;

  HDC hDC;

  switch(uMsg)
  {
    case WM_PAINT:
      hDC = BeginPaint(hWnd, &ps);
      TextOut(hDC, 100,100,
        "Hola, desde el API Win32 para Windows 95/NT", 43);
      EndPaint(hWnd, &ps);
      break;

    case WM_DESTROY:
      PostQuitMessage(0);
      break;

    default:
        return DefWindowProc(hWnd, uMsg, wParam, lParam);
  }
  return 0;
}
```



Aplicaciones a la carta desde AppWizard.

En el listado anterior se puede observar una función donde se atienden a los mensajes *WM_PAINT* y *WM_DESTROY*. Todos los demás mensajes que no son procesados deben de ser reenviados al procedimiento por defecto suministrado por Windows. Hay que resaltar la llamada a la función ***PostQuitMessage()*** dentro del mensaje *WM_DESTROY* y que envía el mensaje *WM_QUIT* a la aplicación, lo que se traduce en la salida del bucle de mensajes y en el final de la aplicación.

Otro mensaje importante es *WM_PAINT*, donde se deben procesar todas las operaciones para pintar sobre la ventana, ya que este mensaje es recibido siempre que la aplicación debe ser redibujada.

El fichero de definiciones

En todos los programas para Windows se debe definir un fichero de definiciones (.DEF), encargado de definir el mode de enlazado de las aplicaciones. Un formato tipo podría ser el siguiente:

```
NAME        Ejemplo
DESCRIPTION   'Programa de ejemplo'

CODE        PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE      1024
STACKSIZE     5120

EXPORTS
          WndProc
```

donde,

NAME es el nombre del fichero de la aplicación.
DESCRIPTION es una descripción de la aplicación.
CODE define el tipo de segmento que tiene la aplicación, normalmente suele ser cargada al principio (PRELOAD), movible (MOVEABLE) y paginable al disco (DISCARDABLE).

DATA indica el tipo de segmento de datos, que normalmente es cargado al principio, movible y con múltiples instancias para cada aplicación.

HEAPSIZE define el tamaño del montón para las reservas dinámicas con malloc.

STACKSIZE define el tamaño de la pila del programa.

MFCs frente al API

¿Qué aporta la programación en C++ mediante las MFCs con respecto a programar directamente en C con el API? Pues la respuesta más contundente parte de la base del rendimiento del desarrollo y la reutilización posterior de las partes comunes. Si el programador conoce C++ el proceso de aprendizaje de programar para Windows es menor mediante las MFCs que mediante el API, existen herramientas de generación automática de código, se pueden incluir de forma sencilla componentes ya programados y probados, las MFCs encapsulan algunas de las tareas más tediosas de la programación para Windows facilitando así la "vida" de forma notoria...

Cuando se quiere crear una aplicación para Windows utilizando las MFC 4.0, el procedimiento normal pasa por utilizar el *AppWizard*, lo que implica la generación automática de la estructura de la aplicación, frente al proceso tedioso y repetitivo necesario cuando utilizamos el API. Además la integración de los recursos, sobre todo los diálogos, se hace más sencilla y rápida desde las MFC por medio de *ClassWizard* que desde el API, donde hay que realizar de nuevo todo el proceso.

Pero entonces, ¿Por qué hemos visto esta introducción al API y a la creación de una aplicación mínima? La respuesta está en conocer de manera muy básica el funcionamiento e interacción de las aplicaciones con el sistema, y que en la programación con las MFCs se implementa igual aunque quede ocultado por los métodos de programación orientado a objetos.

Próximo artículo

En el próximo artículo entraremos de lleno en la creación de una aplicación utilizando todas las excelentes herramientas que Visual C++ 4.0 nos ofrece y que nos permitirá desarrollar una aplicación bastante coqueta con una inversión de tiempo relativamente pequeña.

Cuando se quiere crear una aplicación para Windows utilizando las MFC 4.0, el procedimiento normal pasa por utilizar el AppWizard

Notas finales

Quiero indicar para los que disponen de conexión a Internet un par de direcciones donde se puede encontrar información sobre Visual C++ 4.0, las MFCs y todo lo relacionado con Microsoft y sus herramientas. Estas direcciones son:

WEB:
http://www.microsoft.com
WEB:
http://www.msn.com

Además y si disponéis de conexión en la **Microsoft Network (MSN)**, sólo tenéis que realizar una búsqueda incluyendo Visual C++, o C++ o cualquier otro acrónimo relacionado con el compilador o la programación para Windows y encontrareis información para dar y tomar.

CONCURSO SÓLO PROGRAMADORES Y MICROSOFT VISUAL C++ 4.0

Todos los meses Sólo Programadores regala a sus lectores Microsoft Visual C++ 4.0 :

Desde la redacción de Sólo Programadores animamos a todos los lectores a enviarnos todo tipo de preguntas, dudas o comentarios sobre este nuevo curso dedicado a la programación en Visual C++ 4.0 a:

Editorial Tower
Sólo Programadores
ref " Concurso Visual C++ 4.0 "
C/ Aragoneses, 7
28100 Alcobendas. Madrid

Todos los meses se elegirá una carta entre todas las recibidas valorándose el nivel de conocimientos, originalidad, etc. y se obsequiará a su autor con esta magnífica herramienta.

¿AÚN NO HA COMPRADO MICROSOFT® WINDOWS 95?
¿Sabe que con Microsoft® Windows 95 ya puede hacer lo que siempre soñó?
✓ Guardar sus documentos con nombre más largo.
✓ Otras tareas mientras imprime como guardar, copiar, pegar, etc.
✓ Conectarse a redes internacionales (Internet, Microsoft Network, Compuserve...)
Pídalo ya en los Centros de Actualización Microsoft o en su distribuidor habitual.
Para más información llámenos al telf.: (91) 804 00 96

Microsoft
¿HASTA DONDE QUIERES LLEGAR HOY?
PROG.

RESULTADO DEL 1ER TORNEO DE DEMOS E INFOGRAFÍA

Por primera vez en la historia de nuestra publicación se ha celebrado un torneo dedicado a todos nuestros lectores fascinados por el mundo de la demoscene. A todos ellos, "esos locos" que sueñan con líneas de ensamblador, optimizaciones de código, crear la rutina más espectacular y rápida jamas vista. Ellos, los programadores y también los gráfistas o artistas del pixel, ya que sin ellos jamás los primeros podrían plasmar sus ideas de una forma tan esbelta, y conseguir hacer más poderosas esas producciones por ellos creadas y por todos conocidos como las demos.

Tras la conclusión de este primer certamen podemos decir con todo orgullo que ha sido un completo éxito, y todo ello gracias a vosotros, quienes desde todos los rincones de España nos habéis enviado vuestras producciones. Algunas de ellas nos han sorprendido muy gratamente, pero vayamos por categorías comentando un poco.

PREMIOS

ISPANIA	125000	PTAS	
ISPANIA	31250	"	+ Pack biblioteca profesional Tower
MARATHON MAN	31250	"	+ Pack biblioteca profesional Tower
XIMO C.'95	31250	"	+ Pack biblioteca profesional Tower
XIMO C.'95	31250	"	+ Juegos DDM

En la categoría de infografía nos hemos encontrado imágenes tan espectaculares como 1mano.tga, Habita.jpg o también las realizadas por grafista XIMO C.'95, siendo el más destacado en las categorías de imagen fija y animación con Combate.jpg y otras producciones dotadas de una gran calidad e imaginación, sencillamente son espectaculares, no tiene otra palabra. Pero para espectaculares las producciones realizadas en las categorías de demos, donde se ha alcanzado mayor nivel, con algunas demos e intros que no tienen nada que envidiar a otras realizadas fuera de nuestras fronteras.

RESULTADOS POR CATEGORÍAS

Categoría	Sección	Autor	Titulo
Demos	Intros 4 Kb	Marathon Man	RC-Mania
	Intros 64 Kb	Ispania	Lal
	Demos	Ispania	Puaj
Infografía	Imagen fija	XIMO C.'95	Combate
	Animación	E. San Pedro	Hey

VAYAMOS POR CATEGORÍAS:

En la categoría de Intros de 4 Kb hemos recibido algunas muestras de la cantidad y calidad de efectos que se pueden conseguir en tan poco espacio, cabe resaltar las producciones Torg, Get-Ready siendo la más destacada en su categoría la realizada por Marathon Man con su intro llamada RC-Mania.

En la siguiente categoría, Intros de 64 Kb, el grupo andaluz Ispania se lleva la palma con su trabajo Lal, seguida muy de cerca por otras intros como My First Intro, realizada por Gorka.

Pero es en la categoría reina de cualquier party que se precie, donde se han visto las más espectaculares demos, donde se han dado cita grupos de renombre en el mundo de la demoscene como The Lord of the Bits, Spanish Lords, Zoran, etc. Siendo estos últimos con su demo Dancing Shadows presentada en la Euskal Party III, una de las destacadas de esta sección. Esta demo con una música excelente, y con unos gráficos como el del Demonio o el de Goku, poco menos que geniales, que hacen ganar muchos puntos su producción en vistosidad. También estuvo en lo más alto compitiendo por el primer puesto la estupenda demo realizada por el grupo Spanish Lords, Simple, con gran cantidad de efectos vistosos y la colaboración de Noisy Man, músico de Iguana, y uno de los mejores músicos de España, con una música dance al nivel de cualquier producción musical actual. Pero The Lord of the Bits no estuvieron muy atrás con su excelente trabajo Geminis, en la que nos muestran autenticas "virguerías" en 3D, siendo con diferencia la más espectacular en este apartado frente a sus rivales, con unos gráficos 3D con sombreados Phong de lo mas realistas. Pero todavía queda una gran demo en el tintero, la ganadora de este primer torneo, la realizó el grupo Ispania, destacando no sólo el nivel técnico de la demo, si no también el diseño y coordinación de todos los elementos que se requieren para una buena realización, efectos, gráficos y sonido, perfectamente combinados y aderezados con un ligero toque humorístico convierten a esta demo en la mejor y triunfadora de nuestro concurso. Felicitaciones para ellos.

Es obligado comentar que la elección no ha sido fácil, ya que, si bien al principio se recibían demos con cuentagotas, al final los envíos acabaron por inundar la redacción. Elegir ganadores nunca es fácil, aunque suene a tópico es verdad, pero es que el nivel mostrado por algunas de las demos es realmente alto. Parece que siempre se dice lo mismo, que si todas eran buenas, que si no sabíamos cuál elegir, etc. No es hablar por hablar, vosotros mismos podréis juzgarlas en un próximo CD-ROM donde se incluirán vuestros trabajos. Seguro que alguno no estaréis de acuerdo con nuestro jurado, pero eso pasa también en todas las parties.

Creemos que podemos concluir nuestro primer certamen calificándolo como todo un éxito, lo cual sin es mérito de nuestros lectores-coders que nos han enviado sus trabajos, sinceramente valoramos todos, puesto que incluso con el peor habremos conseguido nuestro propósito: poner a la gente a programar.

En Sólo Programadores comenzamos a pensar en nuevo certamen , que quien sabe, quizás nos veamos esta vez en persona, al estilo de los grandes parties.

EL SISTEMA DE ARRANQUE EN LINUX

David Aparicio Campos

Desde hace más de un año, en Solo Programadores se ha tratado de impulsar Linux, un sistema operativo multitarea y multiusuario realmente robusto, y con requerimientos de hardware modestos si se les compara con los de muchas aplicaciones típicas hoy día. Esperamos que nuestros lectores ya se hayan familiarizado con la instalación básica de Linux en un PC, de forma que nos dedicaremos hoy a explicar algunos detalles sobre qué hacer para completarla. Las personas que deseen una descripción del proceso de instalación, deberán consultar el apartado de "*contenido de CD-ROM*" del número de Febrero del 96. El libro "*Linux, Unix para todos*" describe la instalación detallada de Linux (Slackware 2.1), y se distribuyó con la revista en Abril del 95.

Pasando ya al tema que nos ocupa hoy, el arranque de Linux, diremos en primer lugar que tenemos tres posibilidades. Todas ellas dan control de una forma u otra al núcleo principal del sistema (el equivalente al *IO.SYS* y *DOS.SYS* de MSDOS), el cual, una vez cargado, monta la partición principal de Linux (denominada *root*, ya que el punto de montaje es /, el directorio raíz del sistema de ficheros) y completa el arranque del sistema.

ARRANQUE CON DISCO FLEXIBLE

El primer método, y mas sencillo, pasa por tener un disco de inicio que contenga el núcleo de Linux. Lo mas básico consiste en aprovechar el mismo disco de arranque de la instalación. Lo único que necesitaremos será indicarle la unidad y partición del sistema de ficheros raíz, cuando se nos pregunte, con un comando como, por ejemplo:

```
mount root=/dev/hda2
```

Indicando así que la partición principal de Linux se encuentra en el primer disco IDE, segunda partición. Si no se indica nada en este comando, el disco de arranque Slackware continuará la carga y pedirá insertar un segundo disquete, (*root*) y se entra en el proceso tradicional ya conocido por todos.
Sin embargo, la creación de un disco de arranque ya configurado es preferible si vamos a utilizar este método con frecuencia. Casi al final de la instalación, en el paso de configuración, *setup* nos invita a la creación del mismo. Como en este punto se conoce la partición donde se encuentra Linux, el disco creado permitirá ejecutar Linux simplemente reiniciando el PC con este disco insertado.

ARRANQUE DESDE MSDOS

El segundo método usa el programa LOADLIN. Se trata, básicamente, de un programa de MS-DOS que se encarga de cargar el núcleo Linux (que también será un fichero de MSDOS) a memoria, y ejecutarlo, pasando como parámetros algunos datos necesarios, como el lugar donde se encuentra la partición principal de Linux.

La instalación de este programa se realiza en varios pasos. Si hemos seleccionado esta opción de la serie A, *setup* nos habrá dejado un fichero comprimido en */mnt/root/loadlin*.zip*. Antes de rearrancar, deberemos copiar éste y la misma imagen del núcleo (*/mnt/vmli-*

En el CD de Febrero, se ha ofrecido la última distribución Slackware de Linux, en su versión 3.00. Este mes comentaremos cómo poner a punto ciertos aspectos de la misma

nuz) en MSDOS. Esto se realiza, bien mediante un disco flexible MSDOS, o bien montando la partición DOS del disco duro y copiando los ficheros directamente. En el primer caso (supuesta la unidad A: para realizar la operación), la secuencia
de comandos podría ser:

```
mkdir /a
mount -t msdos /dev/fd0 /a
cp /mnt/root/lo* /a
cp /mnt/vmlinuz /a
umount /a
rmdir /a
```

Hemos de puntualizar que, la situación de los ficheros no coincide en el proceso de instalación y durante el uso normal de Linux.

Cuando estamos en la fase de *setup*, el sistema "root" es un disco RAM creado a partir del segundo disco de instalación. Es decir, el sistema de ficheros del directorio / es el disco RAM, y la partición del disco duro donde se está creando Linux se encuentra colgando del directorio */mnt*.

Esta situación varía cuando el sistema ya está instalado, puesto que la partición dedicada a Linux pasa a montarse directamente en el directorio raíz. Observe este detalle en la secuencia de comandos anteriormente descrita. Cuando rearranque en Linux en el futuro, el sistema de ficheros de la partición principal colgará de / y no de */mnt*.

Volviendo al cargador, una vez copiados los ficheros necesarios para ejecutar LOADLIN, se reinicia en MSDOS, y se descomprime loadlin*.zip. Entre los ficheros que obtendremos, se enumeran los integrantes básicos del arranque:

- BIOSINTV.SYS: Deberemos cargar este fichero en el arranque de MSDOS, incorporando poniendo esta línea (como la primera) en el CONFIG.SYS

```
DEVICE=\path\BIOSINTV.SYS
```

- LOADLIN.EXE: Este programa es el que, una vez invocado, cargará el núcleo en memoria desde DOS, le pasará algunos parámetros, y le dará control.
- LINUX.BAT: Este fichero de proceso por lotes contiene una línea que invoca a LOADLIN con los parámetros adecuados para arrancar el núcleo. Se deberá modificar esta línea con respecto a la configuración de su PC, indicando la ruta donde tiene el fichero del núcleo (VMLINUZ) que ha debido copiar de Linux a MSDOS, así como la partición donde esta el sistema Linux.

Si se equivoca en algún punto de este proceso, deberá reiniciar el sistema en Linux (usando el disco de arranque de la instalación, tal y como se ha explicado anteriormente) y copiar los fichero adecuados. Recuerde que si no utiliza el segundo disco de la instalación, y monta la partición Linux del disco duro, el sistema de ficheros colgará de /, y no de */mnt*.

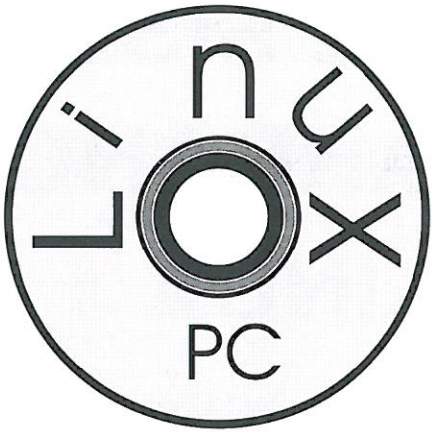
LOADLIN tiene la ventaja de poder arrancar Linux con el sistema UMSDOS, el cual consiste en realizar la instalación en un subdirectorio del sistema de ficheros MSDOS, en lugar de reparticionar el disco duro. Por otro lado, el inconveniente es que este método sólo funciona en DOS, y no en OS/2 ni en Win95.

ARRANQUE CON UN CARGADOR

La tercera forma de inicar Linux es LILO. Se trata de un pequeño cargador que se instala en el sector de arranque y que permite elegir, sobre la marcha, una de varias opciones para activar imágenes de Linux, DOS, o Win95. El arranque de particiones OS/2 es algo diferente, como veremos mas adelante. Durante el setup, se llega a la pantalla de configuración de LILO. Los pasos obligatorios que se permiten aquí son:

1. Comenzar una nueva configuración de LILO
2. Añadir arranque de una partición Linux
3. Añadir arranque de una partición DOS
4. Instalar el sector de arranque de LILO

Una vez realizados estos pasos, habrá quedado en nuestro disco el fichero */mnt/etc/lilo.conf* (durante el setup; */etc/lilo.conf* en el funcionamiento normal), con todos los datos de



la configuración elegida. Tenemos un ejemplo, en la tabla 1. Como podemos observar, se dispone de tres arranques diferentes.

- La sección global indica que se instala LILO en el MBR del disco 0, y que hay una pausa de 5 segundos para poder elegir una opción.
- Cada sección de arranque Linux indica la imagen de núcleo que se carga, cuál es la partición que se monta como root, y con qué etiqueta se denomina a cada opción. Podemos observar en el ejemplo que también se puede arrancar cada partición de Linux con varios núcleos diferentes.
- La sección de arranque 3 es común a todos los sistemas ajenos a Linux, e indica la partición donde se encuentra dicho sistema, la etiqueta con que se denomina, y el lugar donde se encontrará la tabla que describe dicha partición.

Una vez construido dicho fichero de configuración, tecleamos el comando

```
lilo
```

Con esto se "compila" la configuración y se instalará el sistema de arranque. En el ejemplo, el sector de arranque de LILO se graba en un lugar denominado MBR, o sector maestro. Éste es único en cada disco, y es cargado automáticamente por la BIOS tras encender el PC. El inconveniente de este procedimiento es que otros sistemas operativos también pueden instalar su arranque en la MBR, borrando a LILO. Para evitar esto, el cargador se puede instalar en el primer sector de la misma partición Linux, lo cual es menos intrusivo. Por ejemplo, con OS/2 instalado, esto último es imprescindible puesto que el

boot manager necesita arrancar desde la MBR. La segunda línea en *lilo.conf* pasará a ser similar a:

boot=/dev/hda3

Supuesto instalado Linux en la tercera partición. Este método también sirve para particiones lógicas residentes en la extendida. Para que LILO arranque en esta configuración, se deberá marcar dicha partición como "activa" (con fdisk), siendo esta la única diferencia con respecto al arranque de la MBR.
Para desinstalar (o desactivar) LILO, si se ha grabado en la partición Linux, basta con marcar como "activa" otra partición. Si por el contrario se ha grabado en el MBR, el siguiente comando MSDOS borrará el sector donde se encontraba LILO:

fdisk /MBR

Por último, es útil comentar que el arranque por defecto (cuando no se selecciona ninguna opción cuando aparece el prompt de LILO) se produce sobre la primera opción que aparezca en el fichero *lilo.conf* (en nuestro ejemplo, nucleo358). Si deseamos, por ejemplo, que se arranque por defecto en DOS desde LILO, bastará con poner esta opción la primera del fichero (en el ejemplo, mover las últimas cinco líneas justo tras la sección general) y reconstruir todo de nuevo con el comando "lilo".

TRAS EL ARRANQUE

Sea cual sea el método elegido, llega un momento en el que Linux monta la partición principal, y se arranca el resto del sistema. Los pasos que se siguen a partir de ese momento son, a grandes rasgos:

- Se ejecuta el fichero */sbin/init*
- Este programa lee la configuración de */etc/inittab* y la ejecuta.
- Se ejecutan, según el nivel de arranque, los scripts del directorio */etc/rc.d*
- Se arrancan, si se encuentran instalados, los componentes del sistema TCP/IP, para admitir acceso de usuarios por red (pseudoterminales).
- Se lanzan el programa /sbin/*getty* por cada línea (de consola o serie) en la que se aceptan usuarios, y el sistema queda esperando los accesos de los mismos.

Tras todo esto, concluye el inicio del sistema. El núcleo disponible ahora será con toda probabilidad un paso atrás con respecto a lo que teníamos durante la instalación. Las unidades de CD, tarjetas de red, y controladoras SCSI pueden haber desaparecido. Esto sucede porque el núcleo activo (*/vmlinuz*) no es el que teníamos en el disco de arranque, sino uno que se encontraba en la serie A, equivalente al disco "bare" o "scsi", según lo que hayamos elegido en *setup*.
Bien se deberá recompilar un núcleo nuevo, tal y como se explicó en un artículo anterior (para lo que deberá tener instalada la serie D), o bien deberemos instalar un núcleo precompilado de la serie Q, los cuales incorporan mayor variedad de drivers.

Para aquellos recién llegados a Unix, haremos una equivalencia (salvando las evidentes distancias) entre los ficheros de configuración del arranque de DOS y de Linux.

El equivalente al CONFIG.SYS de MSDOS se encuentra repartido en varios ficheros dentro del directorio */etc/rc.d*. La explicación de que existan tantos ficheros en este directorio es que Unix tiene varios niveles de arranque, como mono o multiusuario, paso automático a modo X-Windows durante el arranque, y muchos otros detalles (ver "man init")

AUTOEXEC.BAT tiene su igual en */etc/profile*, el cual se ejecuta cada vez que se introduce en el sistema cualquier usuario, y permite inicializar el entorno, dar mensajes de bienvenida, llevar cuenta del tiempo de conexión, y muchas otras utilidades. Adicionalmente, hay otro fichero de este tipo por cada usuario, que puede ser personalizado por el mismo, y que se encuentra en el directorio de entrada. Se denomina *.profile* (nótese que el punto es importante)

CONCLUSIONES

Con este artículo, se ha pretendido explicar algunos aspectos oscuros del arranque en Linux. En Solo Programadores, la primera revista en castellano que ha tratado el sistema operativo Linux, trataremos de ofrecer regularmente cada nueva versión Slackware en CDROM, comentando los aspectos diferenciadores de las mismas. Esperamos que ya se hayan familiarizado con los rudimentos de la instalación de este sistema. Si usted acaba de tomar el primer contacto con Linux, no se desanime por la dificultad inicial. Si se defiende leyendo documentación en inglés, hay un subdirectorio "*docs*" donde se comentan aspectos varios sobre el sistema. También dispone de la sección de correo del lector, o puede consultar al autor de esta sección, por carta a la redacción o por correo e-mail, a daparic@colibri.tid.es

Suerte, y hasta otro mes.

TABLA 1

```
# — Seccion global (comienzo)
boot = /dev/hda
delay = 50
# — Seccion global (fin)
# — Comienzo de la seccion (1) de
arranque en Linux
image = /k358
  root = /dev/hda3
  label = nucleo358
  read-only
# — Fin de la seccion (1)
# — Comienzo de la seccion (2) de
arranque en Linux
image = /vmlinuz
  root = /dev/hda3
  label = viejo
  read-only
# — Fin de la seccion (2)
# — Comienzo de la seccion (3) de
arranque DOS
other = /dev/hda1
  label = dos
  table = /dev/hda
# — Fin de la seccion (3)
```

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

P Desarrollo programas para el DSP32C de AT&T y las aplicaciones que necesito (compilador y enlazador) se deben utilizar desde la línea de comandos del DOS, por lo que tengo que estar saliendo continuamente del editor para compilar, memorizar los errores y volver a editar. Esto es tedioso y resulta muy difícil el recordar los errores. La utilidad que utilizo (d3make) no permite generar un fichero con la información de los errores y no se que hacer para mejorar mi trabajo. Un saludo .

Francisco Javier Diego Martín
(Salamanca)

R Aunque el make que utiliza no disponga de opciones para la generación de un fichero con las salidas de los programas que ejecute (usualmente compiladores y enlazadores), "siempre" está disponible la posibilidad de redireccionar la salida a pantalla a un fichero mediante la opción ">" de casi todos los Sistemas Operativos:
d3make > salida.txt

Esto funcionará siempre y cuando los programadores que desarrollaron el programa utilizan la salida estándar stdout, (suele ser la pantalla por defecto) para enviar su información al usuario. El símbolo > redirecciona la salida estándar a donde se le indique:

```
> prn:
> con:
> fichero
> lpt1:
```

Además de esto, puede buscarse un editor que interprete la salida del programa make y que muestre los errores conjuntamente con el fuente, a la manera de los entornos integrados de desarrollo (Borland, Watcom, etc.) Alguno de los editores pensados para programación, que incluyen esta característica son: Brief o el excelente Boxer.

P ¿Como puedo obtener algún número de la revista en los que se incluía un CD-ROM con el sistema operativo Linux? Creo que eran los números 6 y 13.

Necesito encontrar el número 12, para poder completar mi "colección" de Solo Programadores ¿Cómo puedo conseguirlo?

Juan Carlos Caramé
(Madrid)
Vicente López Llames
(Oviedo)
Israel Peragón Sucilla
(Leganés/Madrid)

R Para conseguir cualquier producto de Sólo Programadores / Tower Communications, se dispone de varios caminos:
Por correo a la siguiente dirección:
TOWER COMMUNICATIONS SRL
C/ Marqués de Portugalete, 10 - Bajo
28027 Madrid
Por fax al siguiente número:
(91) 320 60 72
Llamando al siguiente teléfono:
(91) 741 26 62
Horario de 9 a 14 y de 15:30 a 18:30 horas

P He observado que algunos lectores de CDs musicales muestran el título del compacto, así como el de la canción que se está escuchando. ¿Como podría leer esta información desde un programa desarrollado por mi? Gracias.

Unai Uribarri Rodríguez
(Asturias)

R Los Cds musicales no contienen dicha información. Lo que permiten algunos reproductores de gama alta, es almacenar, de manera totalmente manual, un nombre para el disco, y algunos (muy pocos) hasta un nombre para cada canción. El reproductor identifica el CD mediante la duración total de la grabación y mediante el número de canciones, para así acceder a su base de datos interna y mostrar los textos almacenados. Este sistema es el utilizado también por los programas reproductores que existen para los ordenadores y que utilizan la unidad de CD-ROM como cadena musical. Con ellos se pueden crear verdaderas enciclopedias musicales y que reconozcan cualquier canción que se les pida. Los nuevos soportes digitales para música, como el mini-disk, sí que incluyen los nombres de las canciones y del disco en la grabación original, por lo que no hace falta "picárselos" previamente.

ENTRE EN EL MUNDO DE
SUPEROFERTA PARA SUSCRIPTORES Y LECTORES DE
SÓLO PROGRAMADORES
Revista práctica para usuarios de PC
+ 50% DESCUENTO en VISUAL C++
LA PROGRAMACIÓN PARA Microsoft Windows 95 Y Microsoft WINDOWS NT
Microsoft VISUAL C++
P.V.P. 58.000
PRECIO ESPECIAL
26.900
+ IVA
CON LAS HERRAMIENTAS VISUALES DE MICROSOFT
PARA BENEFICIARSE DE ESTA OFERTA LOS LECTORES NO SUSCRIPTORES DEBERÁN REMITIR LOS CUPONES QUE APARECERÁN EN ESTA PÁGINA
CURSO DE VISUAL C++
CUPÓN 1

SARTECO
96
XXI JORNADAS DE COMUNICACIONES, INFORMATICA Y ELECTRONICA
4-7 DE MARZO
ETSI DE TELECOMUNICACION • CIUDAD UNIVERSITARIA. MADRID
UNIVERSIDAD POLITECNICA DE MADRID ■ COLABORA: FUNDACION UNIVERSIDAD-EMPRESA